

Fig. 2

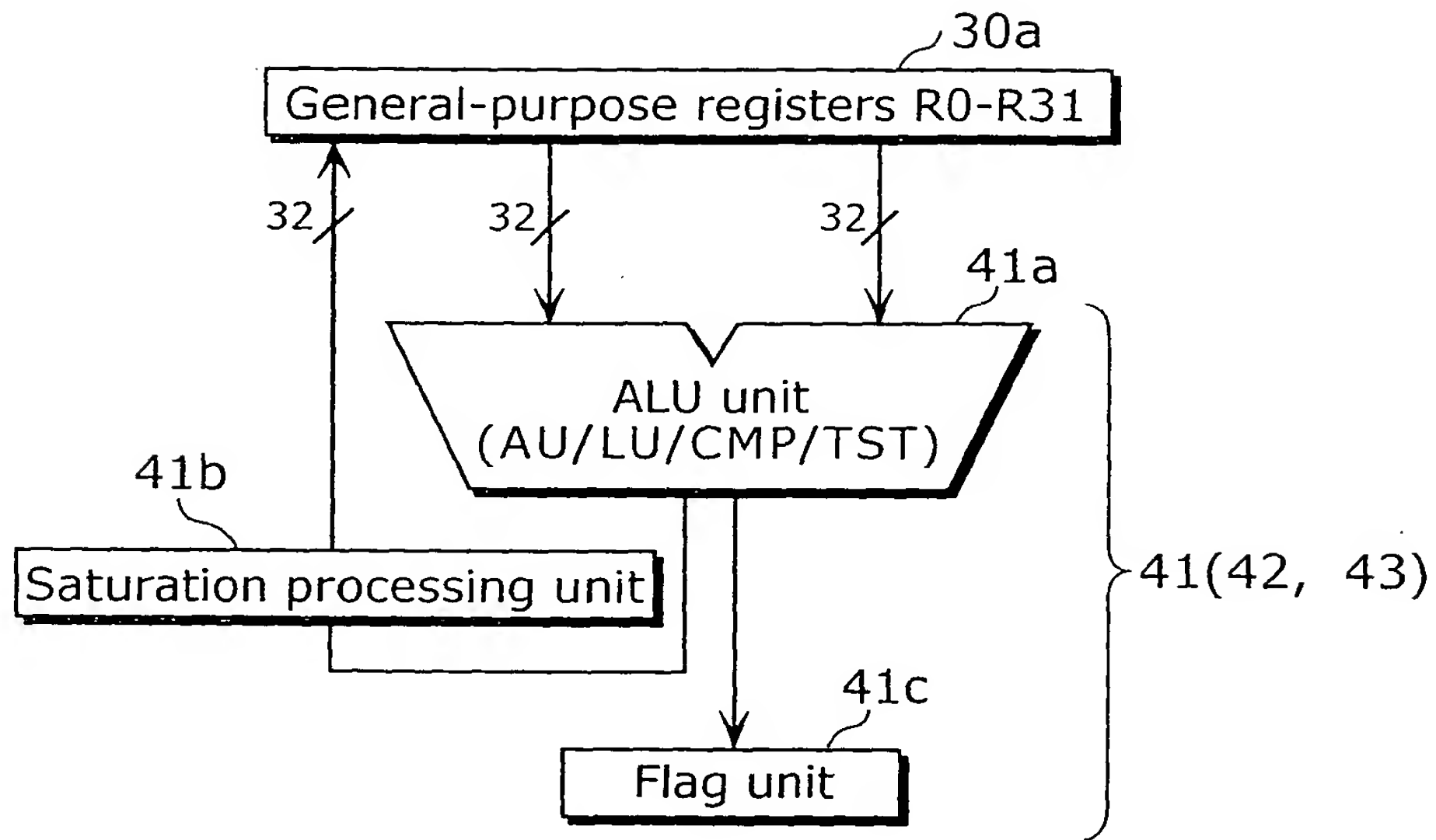


Fig. 3

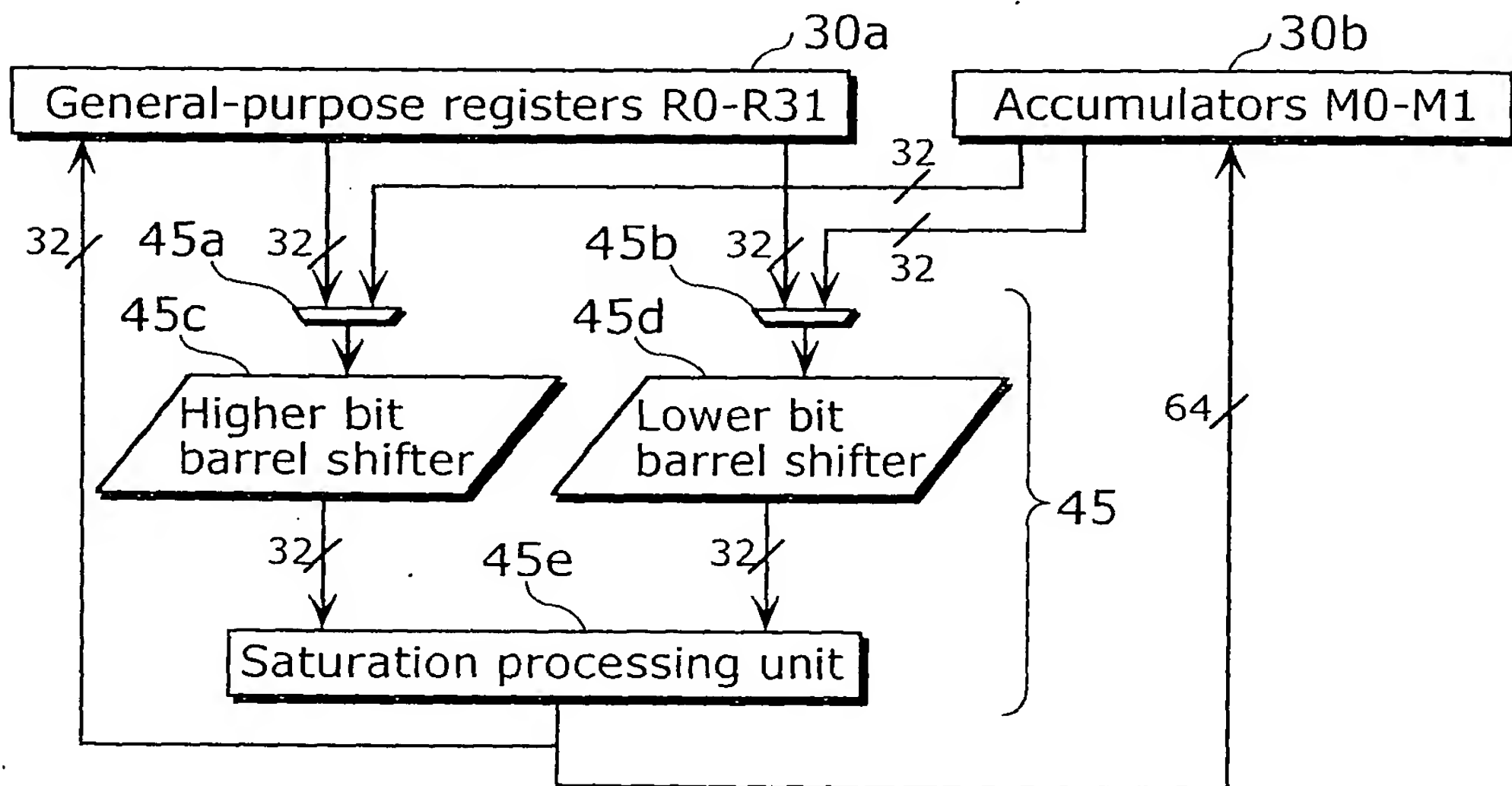


Fig. 4

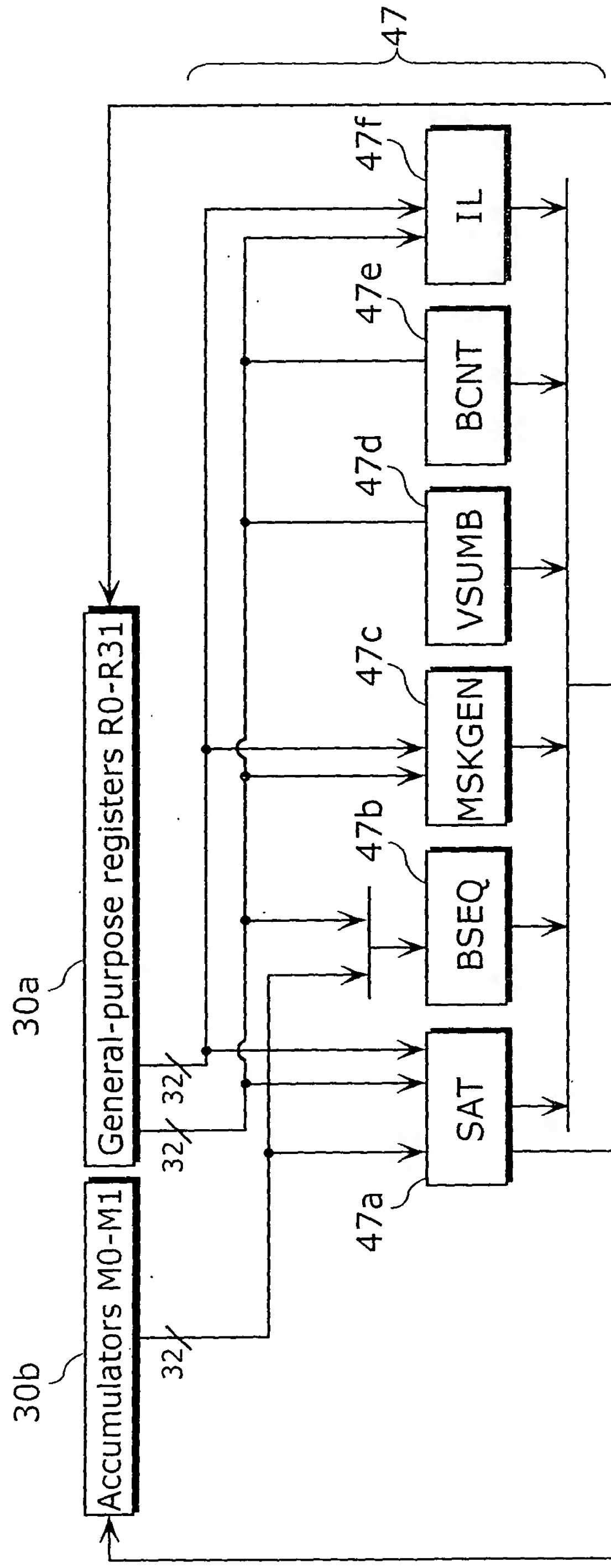


Fig. 5

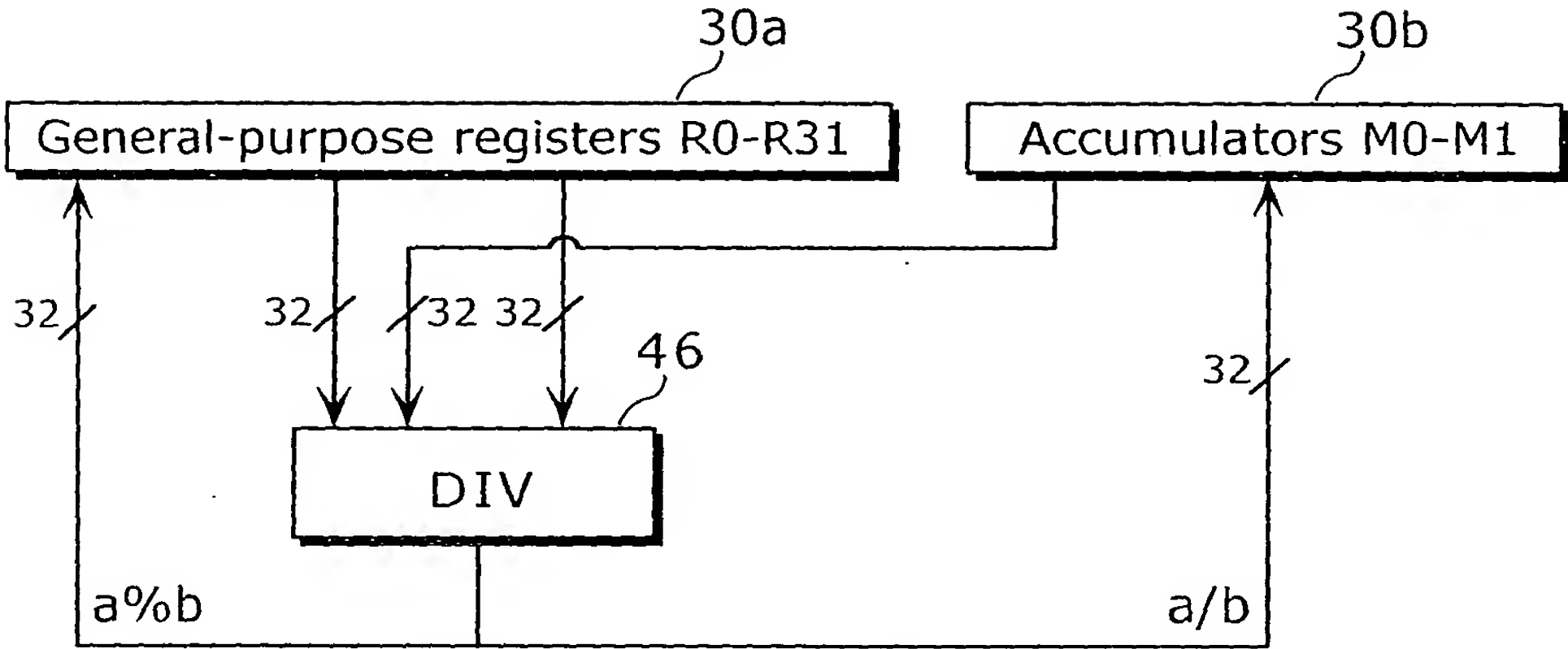


Fig. 6

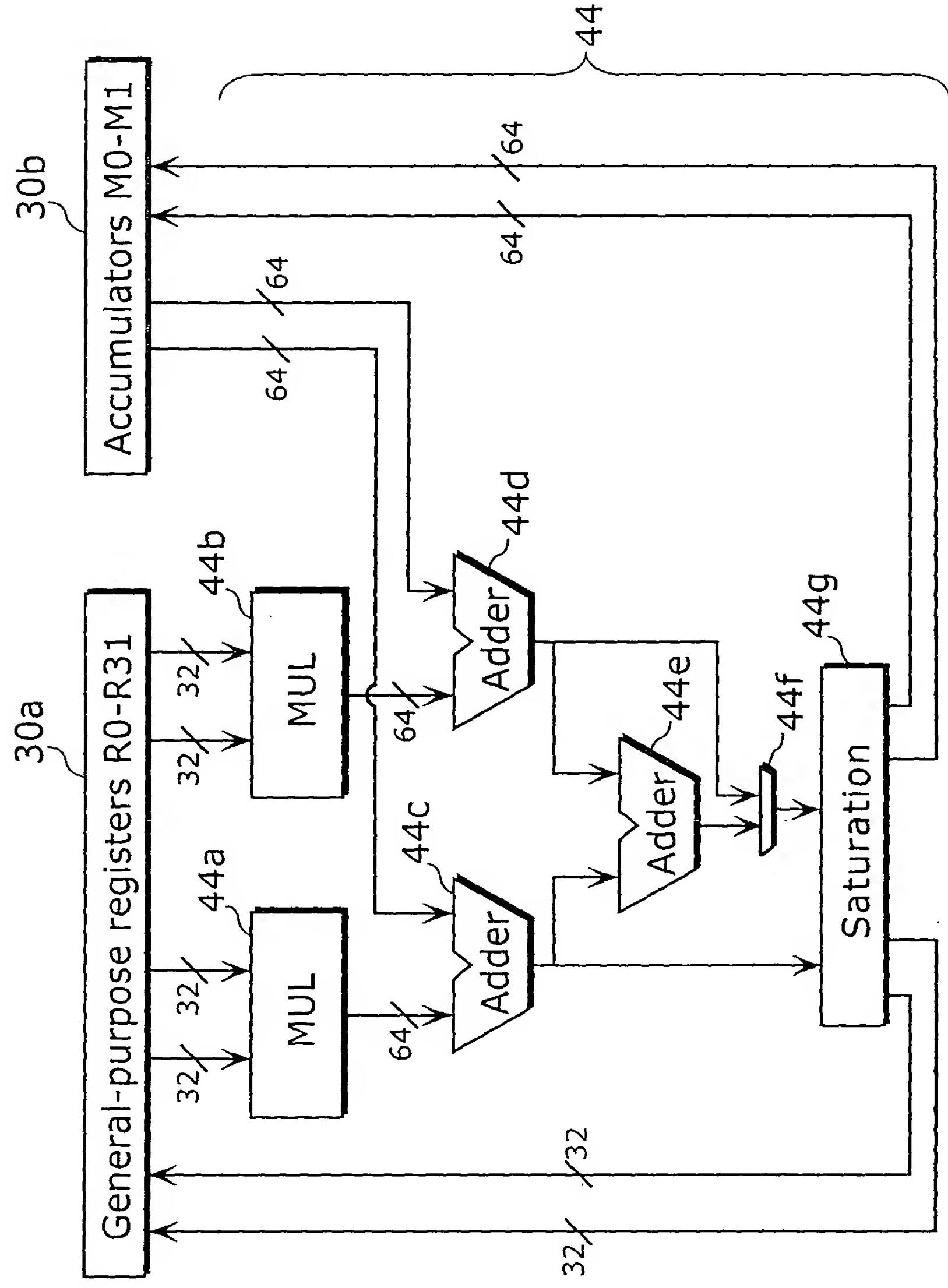


Fig. 7

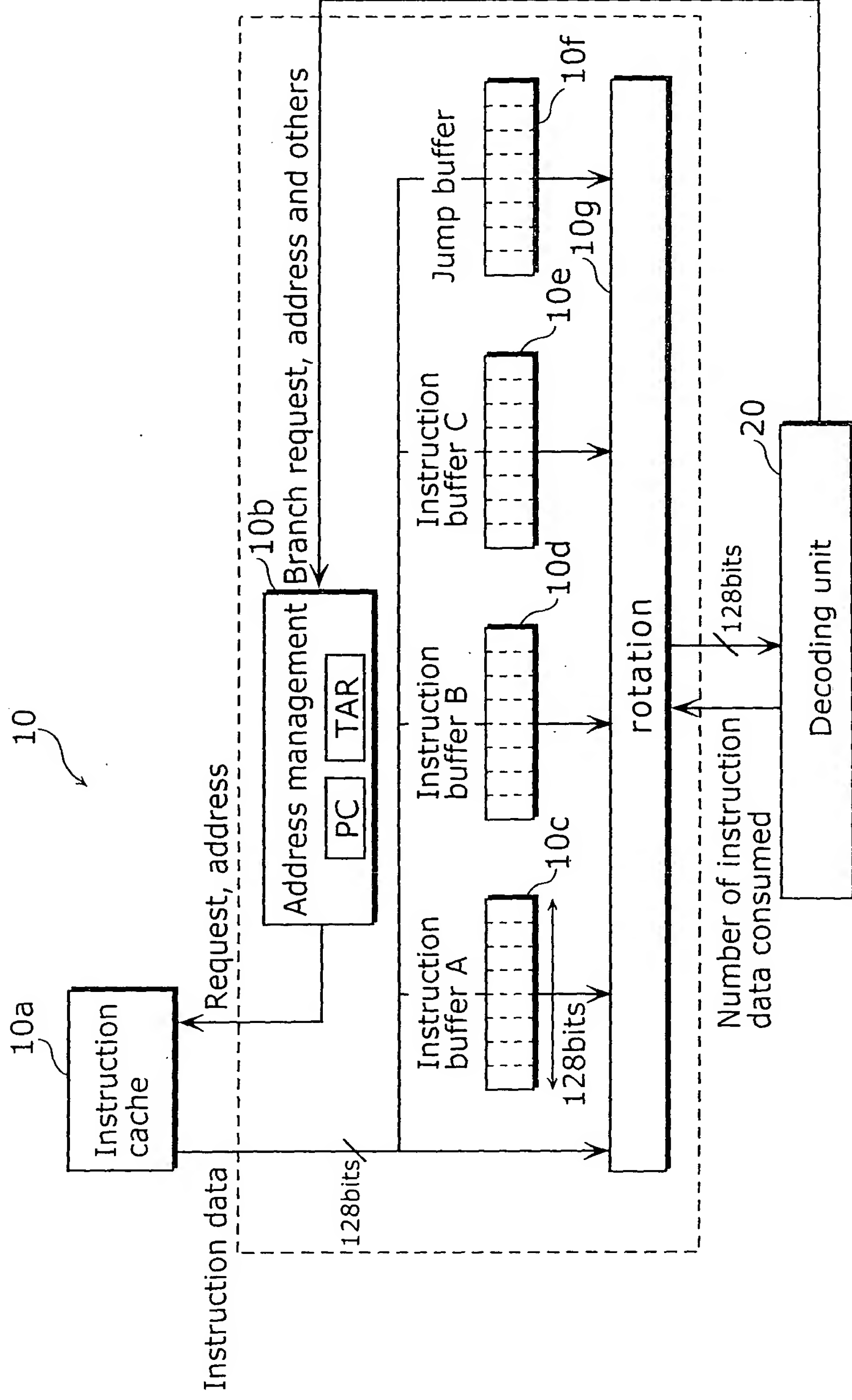


Fig. 8

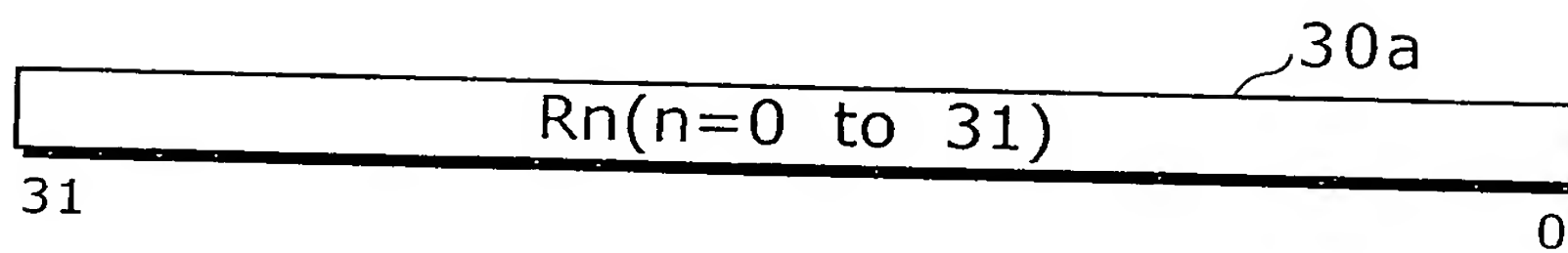


Fig. 9

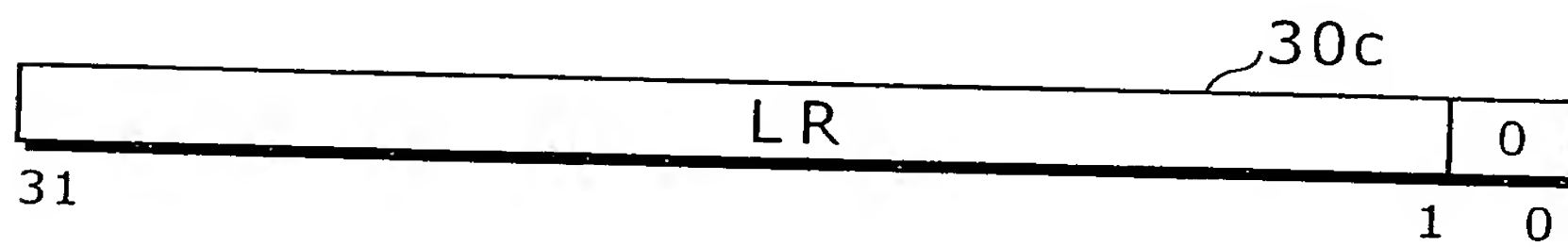


Fig. 10







Fig. 12

32

Bit	31	30	29	28	27	26	25	24
Bit name	ALN		reserved	BPO				
Bit	23	22	21	20	19	18	17	16
Bit name	reserved			VC3		VC2	VC1	VC0
Bit	15	14	13	12	11	10	9	8
Bit name	reserved						OVS	CAS
Bit	7	6	5	4	3	2	1	0
Bit name	C7	C6	C5	C4	C3	C2	C1	C0

Fig. 13A

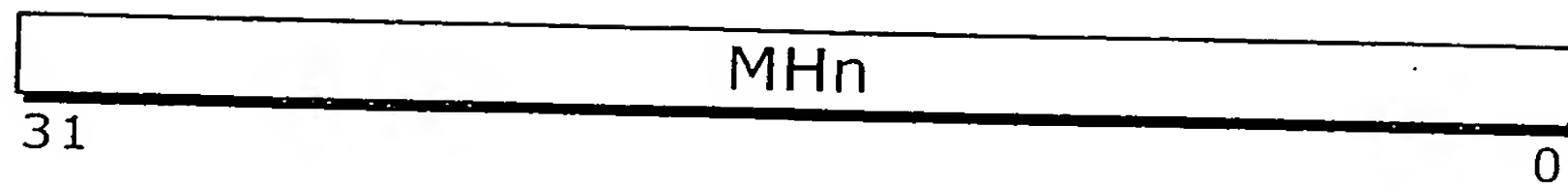
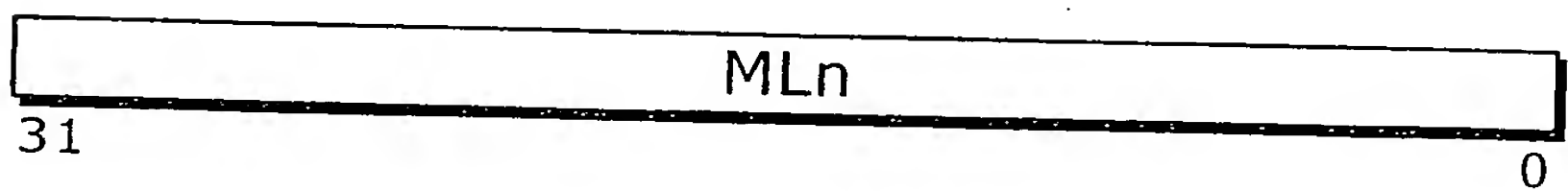


Fig. 13B



30b

Fig. 14

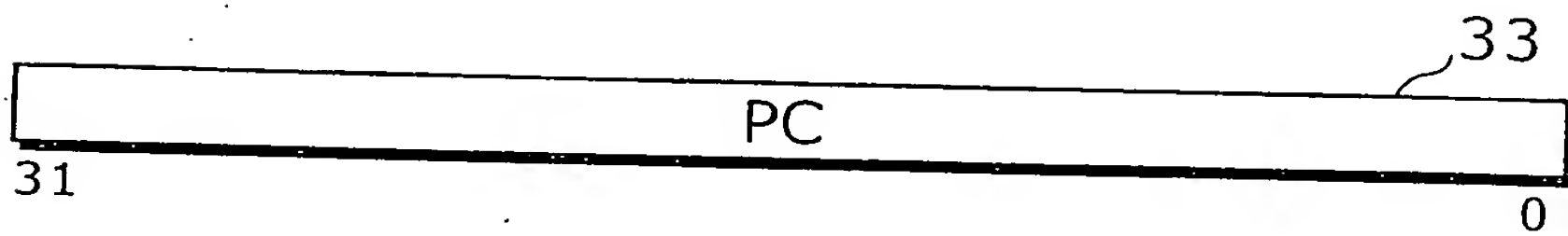


Fig. 15

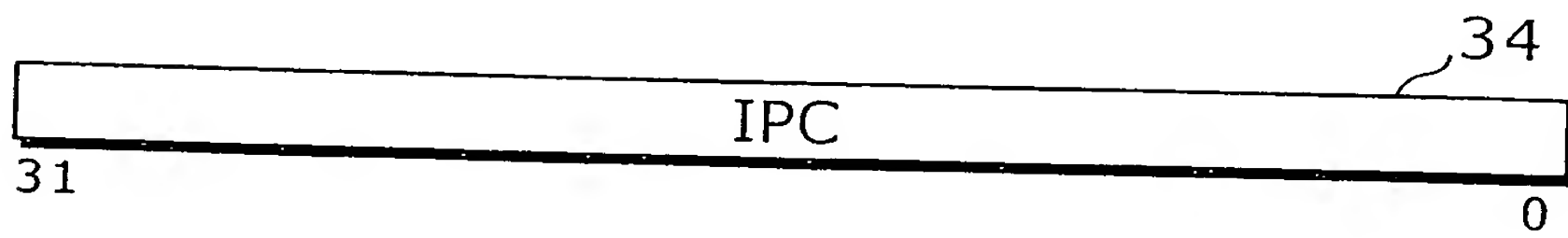


Fig. 16

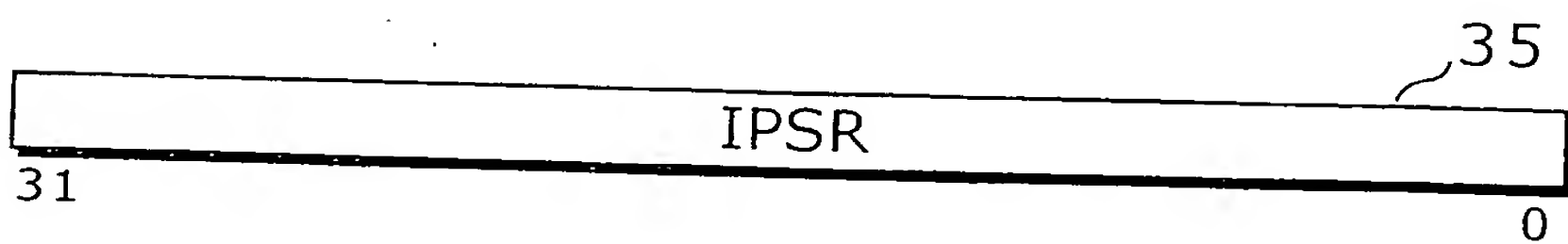


Fig. 17

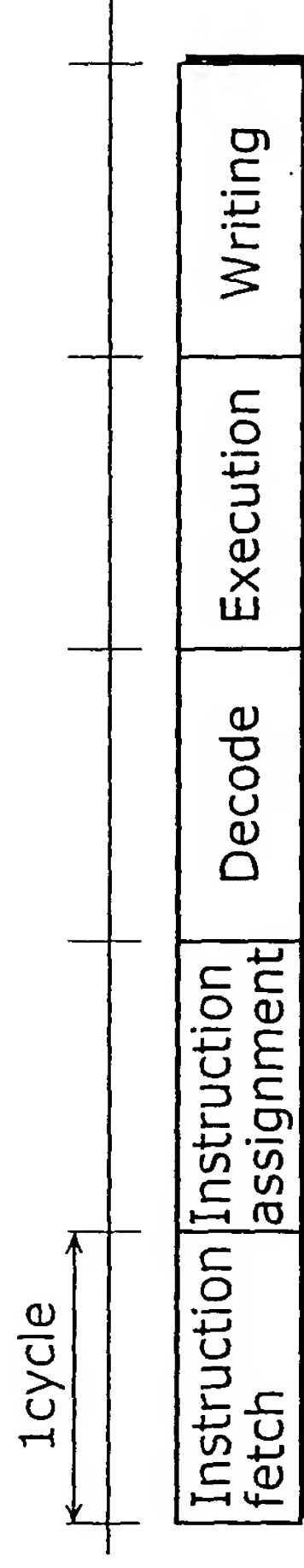


Fig. 18

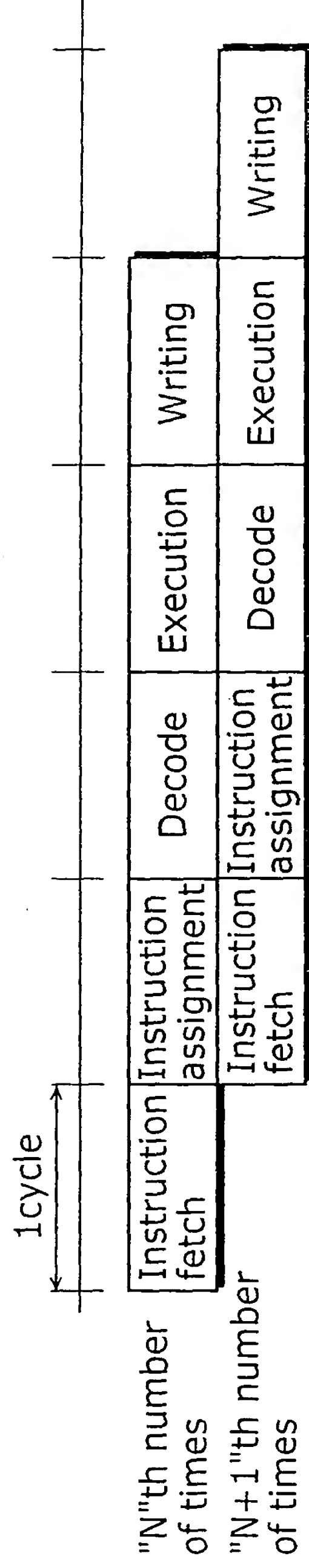


Fig. 19

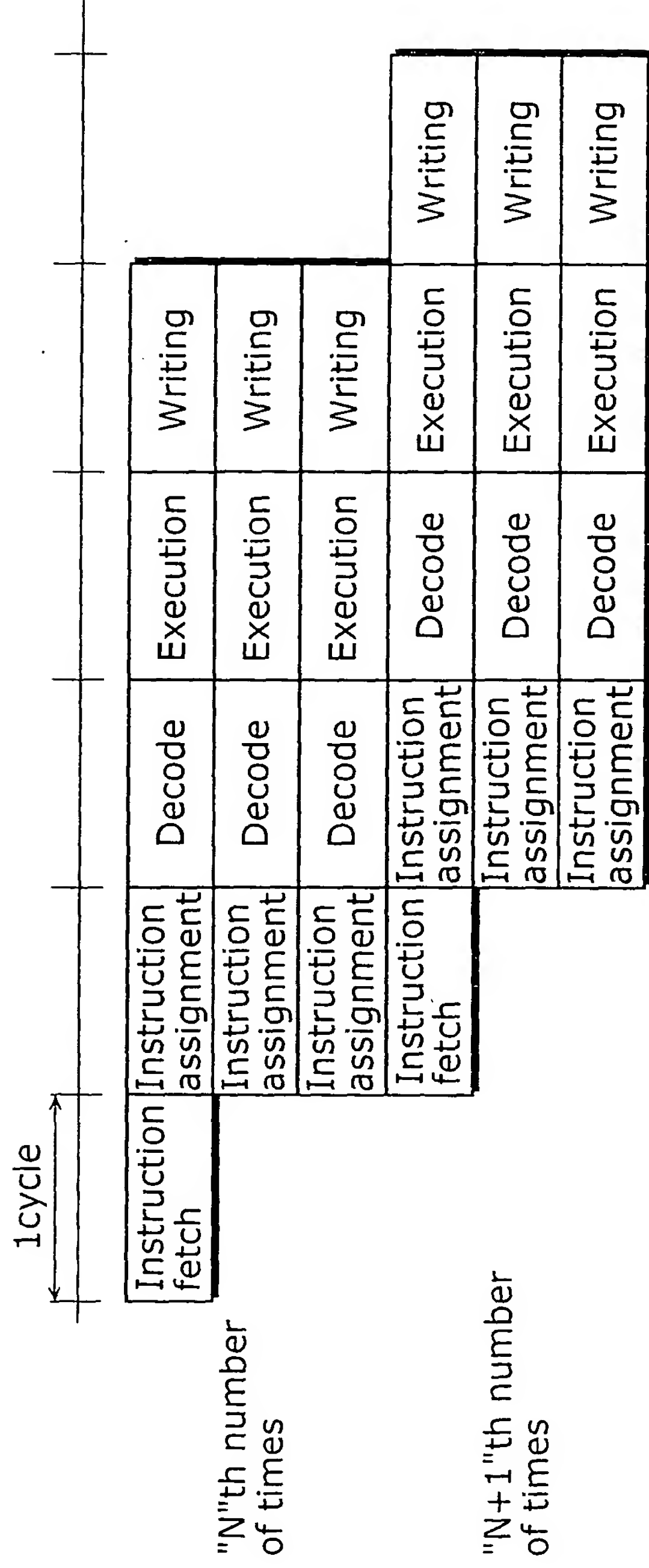


Fig. 20

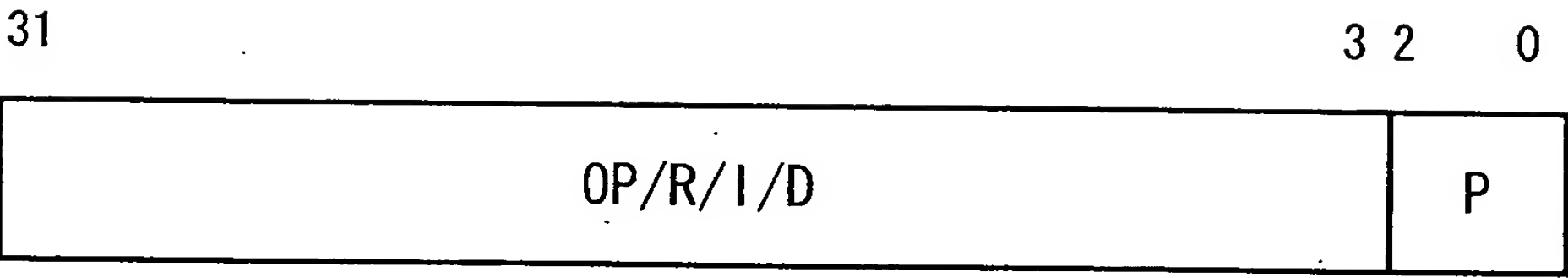


Fig. 21

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
ALU add system	S I N G L E	Word	add	Rc,Ra,Rb Rb,Ra,i12s SP,i19s Ra2,Rb2 Rc3,Ra3,Rb3 Ra2,i05s SP,i11s					32 16
			addu	Rb,GP,i16u Rb,SP,i16u Ra3,SP,i08u					32 16
			addc	Rc,Ra,Rb	W:cas,c0:c1		Addition with carry		
			addvw	Rc,Ra,Rb	W:ovs		Addition with overflow		
			adds	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		32
			addsr	Rc,Ra,Rb			$Ra + Rb + 1 \rightarrow R_c$		
			s1add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + Rb \rightarrow R_c$		16
			s2add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + Rb \rightarrow R_c$ (>>2)		32 16
			addmsk	Rc,Ra,Rb	R:BP0		$Ra + Rb \rightarrow R_c$		
			addarvw	Rc,Ra,Rb					
		Half word	faddvh	Rc,Ra,Rb	W:ovs				
		S I M D	vaddh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vaddhvh	Rc,Ra,Rb	W:ovs		$Ra + Rb \rightarrow R_c$		
			vsaddh	Rb,Ra,i08s			$Ra + Rb \rightarrow R_c$		
			vaddsh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vaddsrh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vaddhvc	Rc,Ra,Rb	R:VC				
			vaddrhvc	Rc,Ra,Rb					
			vxaddh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vxaddhvh	Rc,Ra,Rb	W:ovs		$Ra + Rb \rightarrow R_c$		
			vhaddh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vhaddhvh	Rc,Ra,Rb	W:ovs		$Ra + Rb \rightarrow R_c$		
			viaddh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			viaddhvh	Rc,Ra,Rb	W:ovs		$Ra + Rb \rightarrow R_c$		
		Byte	vaddb	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vsaddb	Rb,Ra,i08s			$Ra + Rb \rightarrow R_c$		
			vaddsb	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		
			vaddsrh	Rc,Ra,Rb			$Ra + Rb \rightarrow R_c$		

Fig. 22

Cate gory	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
ALU sub system	S I N G L E	Word	sub	Rc,Rb,Ra Rb2,Ra2 Rc3,Rb3,Ra3				A	31 16
			rsub	Rb,Ra,i08s Ra2,Rb2 Ra2,i04s			Immediate value - Ra → Rb ( Rb2 )		32 16
			subc	Rc,Rb,Ra	W:cas,c0:c1		With carry		
			subvw	Rc,Rb,Ra	W:ovs		With overflow		
			subvs	Rc,Rb,Ra			Ra - Rb → Rb		
			submsk	Rc,Rb,Ra	R:BP0		Ra CF, BP0 Rb → Rb		
		Half word	fsubvh	Rc,Rb,Ra					
	S I M D	Half word	vsubh	Rc,Rb,Ra			Ra 16 16 - Rb 16 16 → 16 16 Rc		
			vsubhvh	Rc,Rb,Ra	W:ovs				
			vsrsubh	Rb,Ra,i08s			Immediate Immediate value - value - → 16 16 Rb		
			vsubsh	Rc,Rb,Ra			Ra 16 16 - Rb 16 16 → >>1 >>1 16 16 Rc		
			vxsubh	Rc,Rb,Ra			Ra 16 16 - Rb 16 16 → 16 16 Rc		
			vxsubhvh	Rc,Rb,Ra	W:ovs				
			vhsbh	Rc,Rb,Ra			Ra 16 16 - Rb 16 16 → 16 16 Rc		
			vhsbhvh	Rc,Rb,Ra	W:ovs				
			vlsbh	Rc,Rb,Ra			Ra 16 16 - Rb 16 16 → 16 16 Rc		
			vlsbhvh	Rc,Rb,Ra	W:ovs				
		Byte	vsubb	Rc,Rb,Ra			(Immediate value) Ra 8 8 8 8 - Rb 8 8 8 8 → 8 8 8 8 Rc		
			vsrsubb	Rb,Ra,i08s					
			vasubb	Rc,Rb,Ra	RVC				

Fig. 23

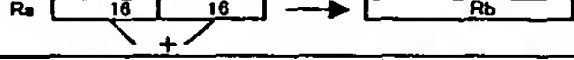
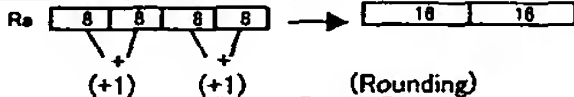
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
ALU logic system	SINGLE	Word	and	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2			AND	A	32
			andn	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2					16 32
			or	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2			OR		16 32
			xor	Rc,Ra,Rb Rb,Ra,i08u Ra2,Rb2			Exclusive OR		16 32
									16
									16
ALU mov system	SINGLE	Word	mov	Rb,Reg32 Reg32,Rb Rb2,Reg16 Reg16,Rb2 Ra2,Rb Ra,i16s Ra2,i08s			Reg32 = TAR LR SVR PSR CFR MH0 MH1 ML0 ML1 EPSR IPC IPSR PC EPC PSR0 PSR1 PSR2 PSR3 CFR0 CFR1 CFR2 CFR3 Reg16 = TAR LR MH0 MH1	A	32
			movp	Rc,Rc+1,Ra,Rb			Rc←-Ra; Rc+1←-Rb;		16
			movcf	Ck,Cj,Cm,Cn			Ci←-Cj; Cm←-Cn;		32
			mvclvovs	Cm:Cm+1	W:ovs		Cm:Cm+1←-CFR.OVS; CFR.OVS; CFR.OVS←-0;		16
			mvclcas	Cm:Cm+1	W:cas		Cm:Cm+1←-CFR.CAS; CFR.CAS; CFR.CAS←-0;		16
			sethi	Ra,i16s					32
			max	Rc,Ra,Rb	W:c0:c1		Rc ← max(Ra,Rb)		
			min	Rc,Ra,Rb	W:c0:c1		Rc ← min(Ra,Rb)		
ALU max min system	SIMD	Word						A	32
		Half word	vmaxh	Rc,Ra,Rb					
			vminh	Rc,Ra,Rb					
		Byte	vmaxb	Rc,Ra,Rb					
ALU abs system	SIMD	Half word						A	32
			abs	Rb,Ra			Absolute value		
			absvw	Rb,Ra	W:ovs		With overflow		
			fabsvh	Rb,Ra	W:ovs				
ALU neg system	SIMD	Half word						A	32
			negvw	Rb,Ra	W:ovs				
			fnegvh	Rb,Ra	W:ovs				
			vnegvh	Rb,Ra	W:ovs				
ALU sum system	SIMD	Half word						A	32
			vsumh	Rb,Ra					
			vsumh2	Rb,Ra					
			vsumrh2	Rb,Ra			(Rounding)		
ALU o t h e r	SIMD	Byte						C	32
			vabssumb	Rc,Ra,Rb					
ALU o t h e r	SIMD	Byte						C	32
			frndvh	Rb,Ra	W:ovs		Rounding		
			vfndvh	Rb,Mn	W:ovs				
			vsel	Rc,Ra,Rb	R:VC				
ALU o t h e r	SIMD	Byte						C	32
			vsgnh	Rb,Ra					



Fig. 24

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
CMP	S I N G L E		cmpCCn	Cm,Ra,Rb,Cn Cm,Ra,i05s,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, le, lt, leu, leu Cm ← result & Cn; (Cm+1 ← ~result & Cn)	A	32
			cmpCCa	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm ← result & Cn; Cm+1 ← ~(result & Cn);		
			cmpCCo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm ← result   Cn; Cm+1 ← ~(result   Cn);		
			cmpCC	C6,Ra2,Rb2 C6,Ra2,i04s	W:CF		CC = eq, ne, gt, ge, le, lt C6 ← result		16
			tstzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm ← (Ra & Rb == 0) & Cn;  (Cm+1 ← ~(Ra & Rb == 0) & Cn)		32
			tstza	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm ← (Ra & Rb == 0) & Cn; Cm+1 ← ~((Ra & Rb == 0) & Cn);		
			tstzo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm ← (Ra & Rb == 0)   Cn; Cm+1 ← ~((Ra & Rb == 0)   Cn);		
			tstnn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm ← (Ra & Rb != 0) & Cn;  (Cm+1 ← ~(Ra & Rb != 0) & Cn)		
			tstna	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm ← (Ra & Rb != 0) & Cn; Cm+1 ← ~((Ra & Rb != 0) & Cn);		
			tstno	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm ← (Ra & Rb != 0)   Cn; Cm+1 ← ~((Ra & Rb != 0)   Cn);		
			tstz	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 ← (Ra2&Rb2 == 0)		16
			tstn	C6,Ra2,Rb2 C6,Ra2,i04u	W:CF		C6 ← (Ra2&Rb2 != 0)		
	S I M D	Half word	vcmpCCh vscmpCCh	Ra,Rb Ra,i05s	W:CF		CC = eq, ne, gt, le, ge, lt		32
		Byte	vcmpCCb vscmpCCb	Ra,Rb Ra,i05s	W:CF		CC = eq, ne, gt, le, ge, lt		

Fig. 25


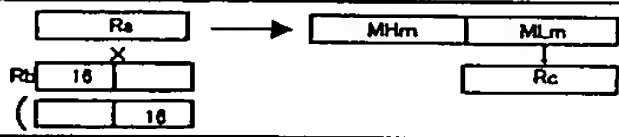
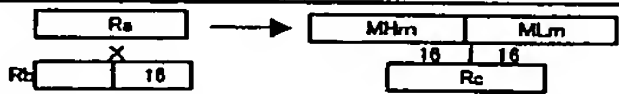
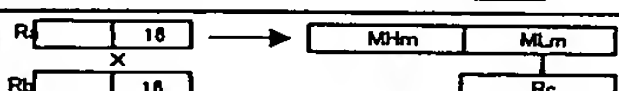
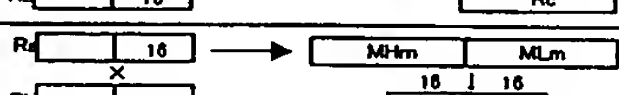
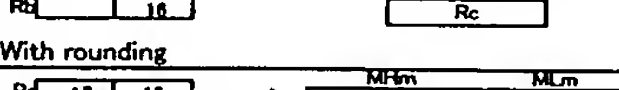
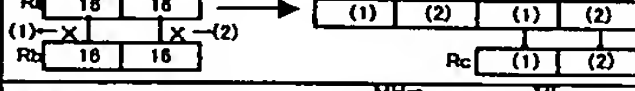
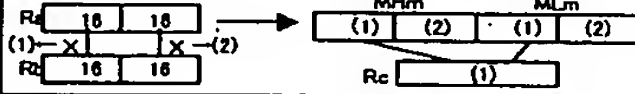
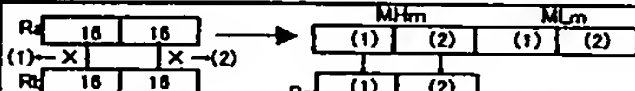
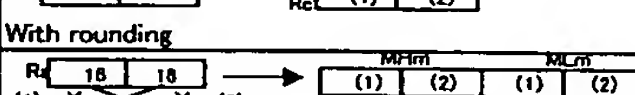
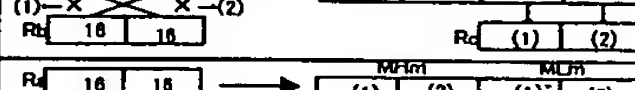
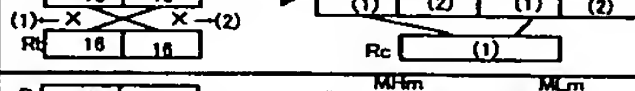
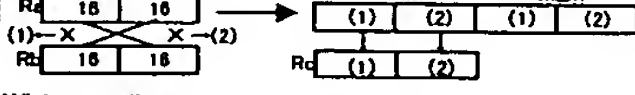
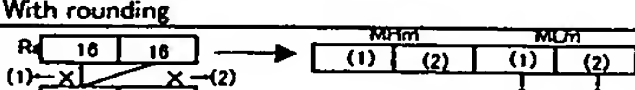
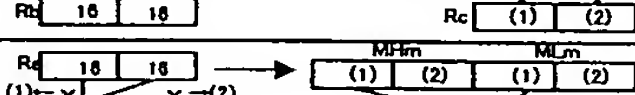
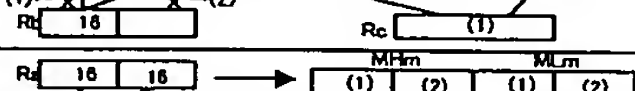
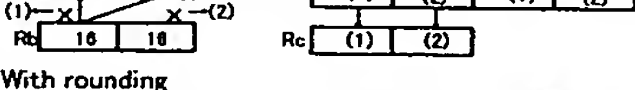
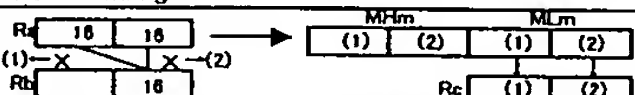
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
mul system	S I N G L E	Word x Word	mul	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s				X2	32
			mulu	Mm,Rc,Ra,Rb Mm,Rb,Ra,i08s			Unsigned multiplication		
			fmulww	Mm,Rc,Ra,Rb		fxp	Fixed point operation		
		Word x Half word	hmul	Mm,Rc,Ra,Rb				X1	
			lmul	Mm,Rc,Ra,Rb					
			fmulhww	Mm,Rc,Ra,Rb		fxp			
		Half word x Half word	fmulhw	Mm,Rc,Ra,Rb		fxp		X1	
			fmulhh	Mm,Rc,Ra,Rb		fxp			
			fmulhhr	Mm,Rc,Ra,Rb		fxp	With rounding		
	vmul		Mm,Rc,Ra,Rb				X2		
	H A L F W O R D x H A L F W O R D	vfmulw	Mm,Rc,Ra,Rb		fxp				
		vfmulh	Mm,Rc,Ra,Rb		fxp				
		vfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding			
		vxmul	Mm,Rc,Ra,Rb						
		vxfmulw	Mm,Rc,Ra,Rb		fxp				
		vxfmulh	Mm,Rc,Ra,Rb		fxp				
		vxfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding			
		vhmul	Mm,Rc,Ra,Rb						
		vhfmulw	Mm,Rc,Ra,Rb		fxp				
		vhfmulh	Mm,Rc,Ra,Rb		fxp				
		vhfmulhr	Mm,Rc,Ra,Rb		fxp	With rounding			
		vimul	Mm,Rc,Ra,Rb						
		vifmulw	Mm,Rc,Ra,Rb		fxp				
		vifmulh	Mm,Rc,Ra,Rb		fxp				
		vifmulhr	Mm,Rc,Ra,Rb		fxp	With rounding			
	Word x Half word	vpfmulhww	Mm,Rc:Rc+1,Ra,Rb Mm,Rc:Rc+1,Ra,Rb		fxp fxp				

Fig. 26

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
mac system	S I N G L E	Word × Word	mac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using mul	X2	32
			fmacww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulww		
		Word × Half word	hmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using hmul	X1	16
			lmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using lmul		32
			fmacww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhww		16
		Half word × Half word	fmacw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhw		32
			fmacw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhw		
			fmacw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhw		
			fmacw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using fmulhw		
	S I M D	H A L F W O R D × H A L F W O R D	vmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vmul	X2	16
			vfmaw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vfmulw		32
			vxmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vxmul		16
			vxfmacw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vxfmulw		32
			vxfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vxfmulh		
			vxfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
		H A L F W O R D × W O R D	vhmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vhmul		
			vhfmacw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vhfmulw		
			vhfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vhfmulh		
			vhfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx					
			vfmaw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Sum of products operation using vfmul		
			vfmaw	Mm,Rc,Ra,Rb,Mn		fxp	Sum of products operation using vfmulw		
			vfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vfmulh		
			vfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
			vlfmach	Mm,Rc,Ra,Rb,Mn Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Sum of products operation using vlfmulh		
			vlfmachr	Mm,Rc,Ra,Rb,Mn Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
		Word × Half word	vpfmacww	Mm,Rc,Rc+1,Ra,Rb,Mn		fxp			

Fig. 27

Cate gory	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
msu  system	S I N G L E	Word × Word	msu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using mul	X2	32
			fmsuww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulww		
		Word × Half word	hmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using hmul	X1	
			lmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using lmul		
			fmsuhww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulhww		
		Half word × Half word	fmsuhw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulhw		
			fmsuhh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using fmulhh		
			fmsuhhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	With rounding		
		H A L F  W O R D  ×  H A L F  W O R D	vmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vmul	X2	
			vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vmul		
	vfmsuh		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vmulh			
	vxmsu		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vxmul			
	vxfmsuw		Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vxfmulw			
	vxfmsuh		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vxfmulh			
	vhmsu		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vhmul			
	vhfmsuw		Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vhfmulw			
	vhfmsuh		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vhfmulh			
	vifmsu		Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			Difference of products operation using vifmul			
	vifmsuw	Mm,Rc,Ra,Rb,Mn		fxp	Difference of products operation using vifmulw				
	vifmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	Difference of products operation using vifmulh				

Fig. 28

Cate gory	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit		
MEM ld system	S I M D	Word	ld	Rb,(Ra,d10u)			<div>Register</div> <div>32</div> <div>←</div> <div>Memory</div> <div>32</div>	M	31	
				Rb,(GP,d13u)					32	
				Rb,(SP,d13u)						
				Rb,(Ra+);i10s						
				Rb2,(Ra2)					16	
				Rb2,(Ra2,d05u)						
		Rb2,(GP,d06u)								
		Rb2,(SP,d06u)								
		Rb2,(Ra2+)								
		Half word	ldh	Rb,(Ra,d09u)			<div>32</div> <div>←</div> <div>16</div> <div></div>		32	
				Rb,(GP,d12u)						
				Rb,(SP,d12u)						
			Rb,(Ra+);i09s							
			Rb2,(Ra2)			16				
			Rb3,(Ra3,d04u)							
		Rb2,(GP,d05u)								
Rb2,(SP,d05u)										
Rb2,(Ra2+)										
Byte	ldb	Rb,(Ra,d08u)			<div>Register</div> <div>32</div> <div>←</div> <div>Memory</div> <div>8</div> <div></div> <div></div>	32				
		Rb,(GP,d11u)								
Byte-> Half word	ldbu	Rb,(Ra,d08u)								
		Rb,(GP,d11u)								
P A I R		ldbh	Rb,(Ra+);i07s			<div>16</div> <div>16</div> <div>←</div> <div>8</div> <div>8</div> <div></div> <div></div>	32			
			Rb,(Ra+);i07s							
		Word	ldp	Rb:Rb+1,(Ra,d11u)				<div>32</div> <div>32</div> <div>←</div> <div>32</div> <div>32</div>	16	
				LR:SVR,(Ra,d11u)						
				TAR:UDR,(Ra,d11u)						
				Rb:Rb+1,(GP,d14u)						
				LR:SVR,(GP,d14u)						
				TAR:UDR,(GP,d14u)						
		Rb:Rb+1,(SP,d14u)								
		LR:SVR,(SP,d14u)								
TAR:UDR,(SP,d14u)										
Rb:Rb+1,(Ra+);i11s										
Rb:Rb+1,(SP,d07u)										
LR:SVR,(SP,d07u)										
Rb2:Re2,(Ra2+)										
Half word	ldhp	Rb:Rb+1,(Ra,d10u)			<div>32</div> <div>32</div> <div>←</div> <div>16</div> <div>16</div>	32				
Rb:Rb+1,(Ra+);i10s										
Rb2:Re2,(Ra2+)										
Byte	ldbp	Rb:Rb+1,(Ra,d09u)			<div>32</div> <div>32</div> <div>←</div> <div>8</div> <div>8</div> <div></div> <div></div>	16				
Rb:Rb+1,(Ra+);i09s										
Byte-> Half word	ldbhp	Rb:Rb+1,(Ra+);i07s			<div>16</div> <div>16</div> <div>←</div> <div>8</div> <div>8</div> <div>8</div> <div>8</div>	32				
ldbuhp	Rb:Rb+1,(Ra+);i07s									

Fig. 29

Cate gory	SIMD	Size	Instruction	Operand	GFR	PSR	Typical behavior	Operation unit	31 16	
MEM store system	S I N G L E	Word	st	(Ra,d10u),Rb (GP,d13u),Rb (SP,d13u),Rb (Ra+);i10s,Rb			Register 32 → Memory 32	M	32	
				(Ra2),Rb2 (Ra2,d05u),Rb2 (GP,d06u),Rb2 (SP,d06u),Rb2 (Ra2+),Rb2			16			
		Half word	sth	(Ra,d09u),Rb (GP,d12u),Rb (SP,d12u),Rb (Ra+);i09s,Rb			16 → 16	M	32	
				(Ra2),Rb2 (Ra2,d04u),Rb2 (GP,d05u),Rb2 (SP,d05u),Rb2 (Ra2+),Rb2			16			
		Byte	stb	(Ra,d08u),Rb (GP,d11u),Rb (SP,d11u),Rb (Ra+);i08s,Rb			8 → 8	M		
		Byte→ Half word	stbh	(Ra+);i07s,Rb			8 8 → 16			
		P A I R	Word	stp	(Ra,d11u),Rb;Rb+1 (Ra,d11u),LR:SVR (Ra,d11u),TAR:UDR (GP,d14u),Rb;Rb+1 (GP,d14u),LR:SVR (GP,d14u),TAR:UDR (SP,d14u),Rb;Rb+1 (SP,d14u),LR:SVR (SP,d14u),TAR:UDR (Ra+);i11s,Rb;Rb+1				32 32 → 32 32	M
					(SP,d07u),Rb;Re (SP,d07u),LR:SVR (Ra2+),Rb2;Re2			16		
			Half word	sthp	(Ra,d10u),Rb;Rb+1 (Ra+);i10s,Rb;Rb+1 (Ra2+),Rb2;Re2			16 16 → 32	32	
								16		
		Byte	stbp	(Ra,d09u),Rb;Rb+1 (Ra+);i09s,Rb;Rb+1			8 8 → 16	M	32	
		Byte→ Half word	stbhp	(Ra+);i07s,Rb;Rb+1			8 8 8 8 → 32			

Fig. 30

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
BRA			setlr	d09s C5,d09s			Set LR Store instruction fetched from LR in branch buffer	B	32
			settar	d09s C6,d09s C6,C2:C4,d09s C6,Cm,d09s C6,C4,d09s	W:c6 W:c2:c4,c6 W:c6,c6 W:c6		Set TAR Store instruction fetched from TAR in branch buffer		16
			setbb	LR TAR			Store instruction fetched from LR in branch buffer Store instruction fetched from TAE in branch buffer		32
			jloop	C5,LR,Ra,i08s C6,TAR,Ra,i08s C6,C2:C4,TAR,Ra,i08s C6,Cm,TAR,Ra,i08s C6,TAR,Ra2 C6,C2:C4,TAR,Ra2 C6,Cm,TAR,Ra2	W:c5 W:c6 W:c2:c4,c6 W:c6,c6 W:c6 W:c2:c4,c6 W:c6		Only predicate [c5] Only predicate [c6]		16
			jmp	TAR LR					32
			jmpI	TAR LR	R:CF				16
			jmpf	TAR LR Cm,TAR C6,C2:C4,TAR					32
			jmpR	LR					16
			br	d20s d09s			Only predicates [c6][c7]		32
			brI	d20s d09s	R:CF				16
			rti			W:PSR R:eh			32
									16
									16

Fig. 31

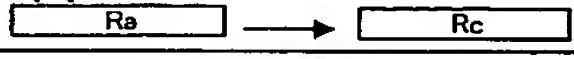
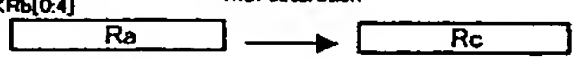
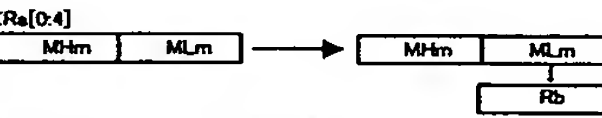
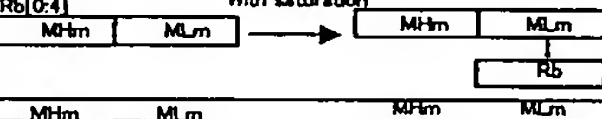
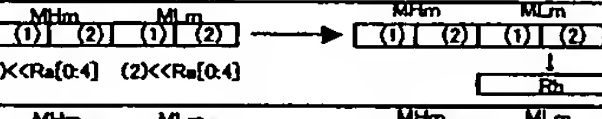
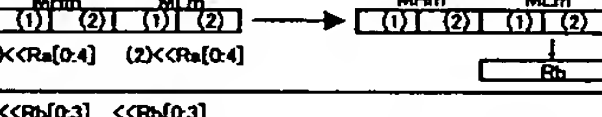

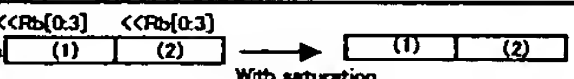
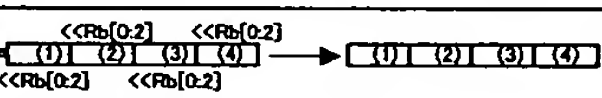

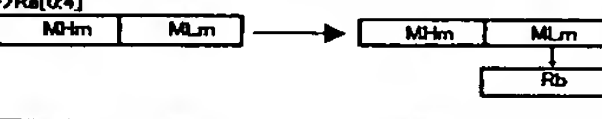
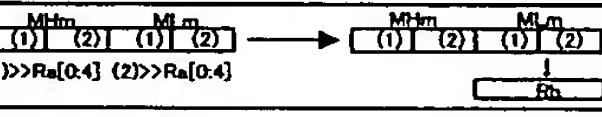

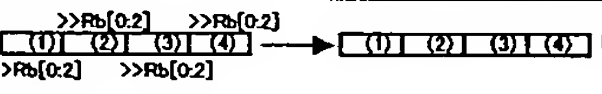
Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	
BS asl system	S I N G L E	Word	asl	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			Left shift <<Rb[0:4] 	S1	32 16
			faslvw	Rc,Ra,Rb Rb,Ra,i05u Rc,Ra,Rb Rb,Ra,i05u	W:ovs		With saturation <<Rb[0:4] 		
		Pair word	aslp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			<<Ra[0:4] 	S2	32
			faslpvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u	W:ovs		With saturation <<Rb[0:4] 		
	S I M D	Word	vasl	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u				S2	
			vfaslvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u	W:ovs				
		Half word	vaslh	Rc,Ra,Rb Rb,Ra,i04u			<<Rb[0:3]   <<Rb[0:3] Ra  Rc	S1	
			vfaslvh	Rc,Ra,Rb Rb,Ra,i04u	W:ovs		<<Rb[0:3]   <<Rb[0:3] Ra  Rc With saturation		
		Byte	vaslb	Rc,Ra,Rb Rb,Ra,i03u					
	BS asr system	S I N G L E	Word	asr	Rc,Ra,Rb Rb,Ra,i05u Ra2,i04u			Arithmetic shift right >>Rb[0:4] 	S1
Pair word			asrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>Ra[0:4] 	S2	32
S I M D		Word	vasr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u					
		Half word	vasrh	Rc,Ra,Rb Rb,Ra,i04u			>>Rb[0:3]   >>Rb[0:3] Ra  Rc	S1	
		Byte	vasrb	Rc,Ra,Rb Rb,Ra,i03u					



Fig. 32

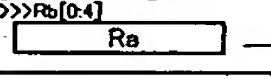
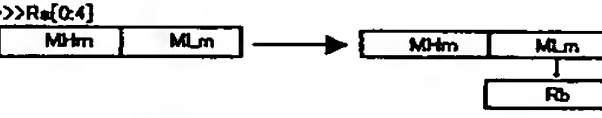
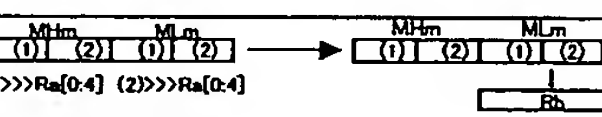
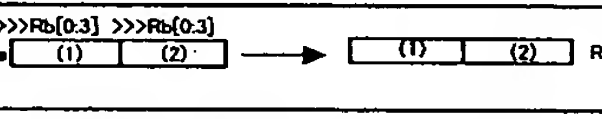
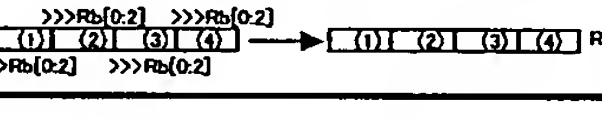
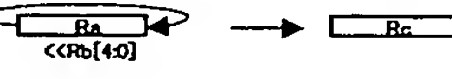
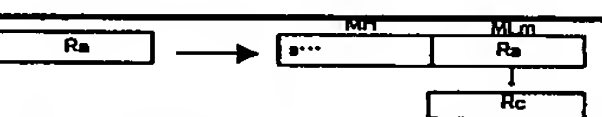

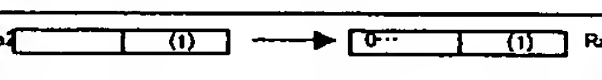
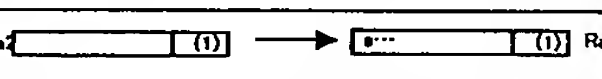
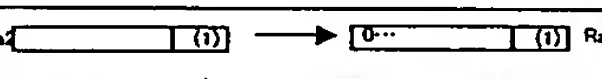
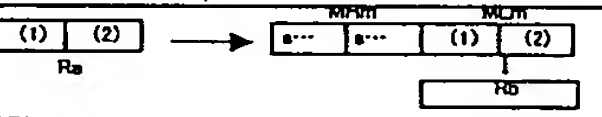
Cate gory	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16	
BS lsr system	S I N G L E	Word	lsr	Rc,Ra,Rb Rb,Ra,i05u			Logical shift right >>>Rb[0:4] 	S1	32	
		Pair word	lsrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>>Ra[0:4] 	S2		
	S I M D	Word	vlsr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u				S1		
		Half word	vlsrh	Rc,Ra,Rb Rb,Ra,i04u			>>>Rb[0:3] >>>Rb[0:3] 			
		Byte	vlsrb	Rc,Ra,Rb Rb,Ra,i03u			>>>Rb[0:2] >>>Rb[0:2] 			
BS rotate system	SIN GLE	Word	rol	Rc,Ra,Rb Rb,Ra,i05u				S1	32	
	S I M D	Half word	vrolh	Rc,Ra,Rb Rb,Ra,i04u						
		Byte	vrolb	Rc,Ra,Rb Rb,Ra,i03u						
BS ext system	S I N G L E	Word	extw	Mm,Rb,Ra				C	32	
		Half word	exth	Ra2				S2		
			exthu	Ra2						
			extb	Ra2						
			extbu	Ra2						
	SI MD	Half word	vexth	Mm,Rb,Ra				C	32	

Fig. 33

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
CNV valn system	SIMD		valn	Rc,Ra,Rb	Raln[1:0]		$\ll (\text{CFR.ALT}[1:0] \ll 3)$ 	C	32
			valn1	Rc,Ra,Rb					
			valn2	Rc,Ra,Rb					
			valn3	Rc,Ra,Rb					
			valnvc1	Rc,Ra,Rb	RVC0				
			valnvc2	Rc,Ra,Rb	RVC0				
			valnvc3	Rc,Ra,Rb	RVC0				
			valnvc4	Rc,Ra,Rb	RVC0				

Fig. 34

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
CNV	SINGLE		bcnt1	Rb,Ra			Count the number of 1s	C	32
			bseq0	Rb,Ra			Count number of values from MSB until first 0 is reached		
			bseq1	Rb,Ra			Count number of values from MSB until first 1 is reached		
			bseq	Rb,Ra			Count number of values from MSB until first -1 is reached		
			mskbrvh	Rc,Ra,Rb	RBP0				
			byterev	Rb,Ra					
			mskbrvb	Rc,Ra,Rb	RBP0				
	SIMD	Half word	vintlh	Rc,Ra,Rb					
			vintlhh	Rc,Ra,Rb					
		Byte	vintlhb	Rc,Ra,Rb					
			vintlhb	Rc,Ra,Rb					
		Half word	vhunpkh	Rb:Rb+1,Ra					
		Byte	vhunpkb	Rb:Rb+1,Ra					
		Half word	vlunpkh	Rb:Rb+1,Ra					
			vlunpkhu	Rb:Rb+1,Ra					
		Byte	vlunpkb	Rb:Rb+1,Ra					
			vlunpkbu	Rb:Rb+1,Ra					
		Half word	vunpk1	Rb,Mn					
			vunpk2	Rb,Mn					
			vstovh	Rb,Ra					
		Byte	vstovb	Rb,Ra					
			vhpkb	Rc,Ra,Rb					

Fig. 35

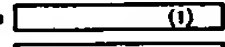
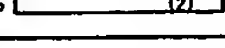
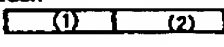
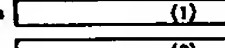
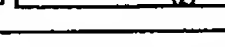
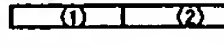
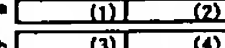
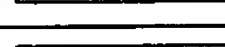
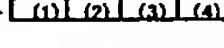
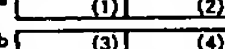

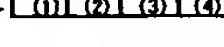

Cate gory	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
SAT vlpk system	S I M D	Word→ Half word	vlpkh	Rc,Ra,Rb			Ra  (1) With saturation Rb  (2) →  (1) (2) Rc	C	32
			vlpkhu	Rc,Ra,Rb			Ra  (1) With unsigned saturation Rb  (2) →  (1) (2) Rc		
		Half word→ Byte	vlpkb	Rc,Ra,Rb			Ra  (1) (2) With saturation Rb  (3) (4) →  (1) (2) (3) (4) Rc		
			vlpkbu	Rc,Ra,Rb			Ra  (1) (2) With unsigned saturation Rb  (3) (4) →  (1) (2) (3) (4) Rc		
SAT sat system	S I N G L E	Word	satw	Mm,Rb,Mn			Word saturation	C	32
			sath	Rb,Ra			Half wordsaturation		
			satb	Rb,Ra			Bytesaturation		
			satbu	Rb,Ra			Unsigned byte saturation		
			sat9	Rb,Ra			9-bit saturation		
			sat12	Rb,Ra			12-bit saturation		
	S I M D	Half word	vsath	Mm,Rb,Mn					
			vsath8	Rb,Ra			Signed 8-bit saturation		
			vsath8u	Rb,Ra			Unigned 8-bit saturation		
			vsath9	Rb,Ra			9-bit saturation		
			vsath12	Rb,Ra			12-bit saturation		

Fig. 36

Category	SIMD	Size	Instruction	Operand	CFR	PSR	Typical behavior	Operation unit	31 16
MSK			mshgen	Rc,Rb Rb,i05U,i05u			Generate mask Rb[12:8] Rb[4:0] 0... 1... 0... or Rb[4:0] Rb[12:8] 1... 0... 1...	S2	32
			mshk	Rc,Ra,Rb Rb,Ra,i05U,i05u			Rb[12:8] Rb[4:0] Rb[4:0] Rb[12:8] 0... 0... or Rb[4:0] Rb[12:8] 0... 0...		
EXTR			extr	Rc,Ra,Rb Rb,Ra,i05U,i05u			Rb[12:8] Rb[4:0] Ra (Without sign extension) Rb (0...)	S2	32
DIV			div divu	MHm,Rc,MHn,Ra,Rb MHm,Rc,MHn,Ra,Rb	W:ovs		Division	DIV	32
ETC			piNI			W:ih,ie,fi,pl R:PSR	Software interrupt N=0~7	B	32
			piN			W:ih,ie,pl R:PSR	Software interrupt N=0~7		16
			scN			W:ih,ie,pl R:PSR	System call N=0~7		
			ldstb	Rb,(Ra)			load bus lock	M	32
			rd	Rb,(Ra) Rb,(d11u) Rb2,(Ra2)		Reee	External register read		16
			wt	(Ra),Rb (d11u),Rb (Ra2),Rb2		Reee	External register write		32
			dpref	(Ra,d11u)			Pre-fetch		16
			dbgmn	i18u			N=0~3	DBGM	
			vcchk		W:CF RVC		VC flag check	B	32
			vmpsw				VMP switching		
			vmpsw	LR					
			vmpintd1			Wie	VMP switching disabled		
			vmpintd2			Wie			
			vmpintd3			Wie			
			vmpinte1			Wie	VMP switching enabled		
		vmpinte2							
		vmpinte3							
		nop				no operation	A	16	

Fig. 37

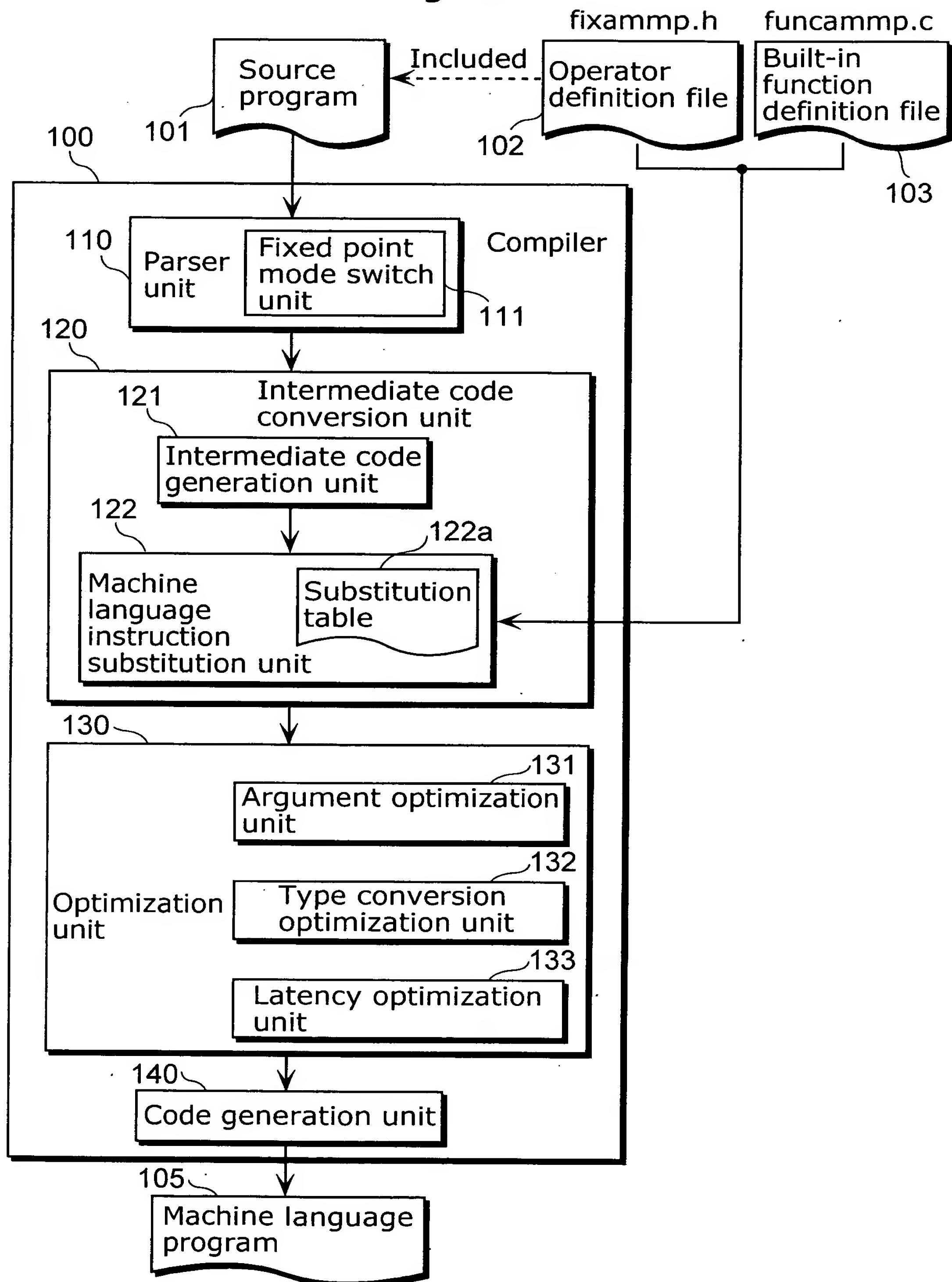


Fig. 38

```

/*****
*
* (C) Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*   fixamp.h
*   Version:
*   Release:
*   Date:      2002/6/14   v0.9.1-convertible
*
*****/

/* Avoid overloading */
#ifndef __FIXAMP__
#define __FIXAMP__

/* FIX-Lib. Class definition */
class FIX16_1;
class FIX32_1;
class FIX16_2;
class FIX32_2;

#ifdef __AMMPCC__
#pragma pack_struct
#endif //__AMMPCC__

////////////////////////////////////
//                               Member of FIX16_1
//
//                               val : Actual value in 16 bits
//
////////////////////////////////////
class FIX16_1 {
    short val;
public:
    // constructor
    FIX16_1() {}
    FIX16_1(int a);
    FIX16_1(float a);
    FIX16_1(double a);
    FIX16_1(FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(volatile FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(const FIX16_1& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_1& operator=(FIX16_1 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_1& operator=(FIX16_1 a)
    {

```

Fig. 39

```

    val = a.val;
    return *this;
}
friend FIX16_1 operator+(FIX16_1 a);
friend FIX16_1 operator-(FIX16_1 a);

friend FIX16_1 operator+(FIX16_1 a, FIX16_1 b);
friend FIX16_1 operator-(FIX16_1 a, FIX16_1 b);

friend FIX16_1 operator*(FIX16_1 a, FIX16_1 b);
friend FIX16_1 operator*(int a, FIX16_1 b);
friend FIX16_1 operator*(FIX16_1 a, int b);
friend FIX16_1 operator*(float a, FIX16_1 b);
friend FIX16_1 operator*(FIX16_1 a, float b);
friend FIX16_1 operator*(double a, FIX16_1 b);
friend FIX16_1 operator*(FIX16_1 a, double b);

friend FIX16_1 operator/(FIX16_1 a, FIX16_1 b);
friend FIX16_1 operator/(FIX16_1 a, int b);
friend FIX16_1 operator/(FIX16_1 a, float b);
friend FIX16_1 operator/(FIX16_1 a, double b);

friend FIX16_1 operator<<(FIX16_1 a, int b);
friend FIX16_1 operator>>(FIX16_1 a, int b);

friend bool operator<(FIX16_1 a, FIX16_1 b);
friend bool operator>(FIX16_1 a, FIX16_1 b);
friend bool operator<=(FIX16_1 a, FIX16_1 b);
friend bool operator>=(FIX16_1 a, FIX16_1 b);
friend bool operator==(FIX16_1 a, FIX16_1 b);
friend bool operator!=(FIX16_1 a, FIX16_1 b);

volatile FIX16_1& operator<<=(int b) volatile;
    FIX16_1& operator<<=(int b);
volatile FIX16_1& operator>>=(int b) volatile;
    FIX16_1& operator>>=(int b);

volatile FIX16_1& operator*=(FIX16_1 b) volatile;
    FIX16_1& operator*=(FIX16_1 b);
volatile FIX16_1& operator*=(int b) volatile;
    FIX16_1& operator*=(int b);
volatile FIX16_1& operator*=(float b) volatile;
    FIX16_1& operator*=(float b);
volatile FIX16_1& operator*=(double b) volatile;
    FIX16_1& operator*=(double b);

volatile FIX16_1& operator/=(FIX16_1 b) volatile;
    FIX16_1& operator/=(FIX16_1 b);
volatile FIX16_1& operator/=(int b) volatile;
    FIX16_1& operator/=(int b);
volatile FIX16_1& operator/=(float b) volatile;
    FIX16_1& operator/=(float b);
volatile FIX16_1& operator/=(double b) volatile;
    FIX16_1& operator/=(double b);
volatile FIX16_1& operator+=(FIX16_1 b) volatile;
    FIX16_1& operator+=(FIX16_1 b);
volatile FIX16_1& operator-=(FIX16_1 b) volatile;
    FIX16_1& operator-=(FIX16_1 b);

```



Fig. 40

```

short value() {return val;}

// Other functions

friend FIX16_1 _fix161(short a);
friend short _bptn(FIX16_1 a);
friend FIX16_1 _fix161(FIX32_1 a);
friend float _float(FIX16_1 a);
friend double _double(FIX16_1 a);

friend FIX16_1 _abs(FIX16_1 a);
friend FIX16_1 _max(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _min(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _adds(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _subs(FIX16_1 a, FIX16_1 b);
friend int _bcnt1(FIX16_1 a);
friend int _bseq(FIX16_1 a);
friend int _bseq0(FIX16_1 a);
friend int _bseq1(FIX16_1 a);
friend FIX16_1 _round(FIX32_1 a);
friend int _extr(FIX16_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_1 a, unsigned int b, unsigned int c);

friend void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mulr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _macr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _msur(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, int b);
friend FIX16_1 _div(FIX16_1 a, float b);
friend FIX16_1 _div(FIX16_1 a, double b);

};

////////////////////////////////////
//                               Member of FIX32_1
//
//                               val : Actual value in 32 bits
//
////////////////////////////////////
class FIX32_1{
    long val;
public:
    // constructor
    FIX32_1() {};
    FIX32_1(int a);

```

Fig. 41

```

FIX32_1(float a);
FIX32_1(double a);
FIX32_1(FIX16_1 a);
FIX32_1(FIX32_1& a)
{
    val = a.val;
}
FIX32_1(volatile FIX32_1& a)
{
    val = a.val;
}
FIX32_1(const FIX32_1& a)
{
    val = a.val;
}

// Operator
volatile FIX32_1& operator=(FIX32_1 a) volatile
{
    val = a.val;
    return *this;
}
FIX32_1& operator=(FIX32_1 a)
{
    val = a.val;
    return *this;
}

friend FIX32_1 operator+(FIX32_1 a);
friend FIX32_1 operator-(FIX32_1 a);

friend FIX32_1 operator+(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator-(FIX32_1 a, FIX32_1 b);

friend FIX32_1 operator*(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator*(int a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, int b);
friend FIX32_1 operator*(float a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, float b);
friend FIX32_1 operator*(double a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, double b);

friend FIX32_1 operator/(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator/(FIX32_1 a, int b);
friend FIX32_1 operator/(FIX32_1 a, float b);
friend FIX32_1 operator/(FIX32_1 a, double b);

friend FIX32_1 operator<<(FIX32_1 a, int b);
friend FIX32_1 operator>>(FIX32_1 a, int b);

friend bool operator<(FIX32_1 a, FIX32_1 b);
friend bool operator>(FIX32_1 a, FIX32_1 b);
friend bool operator<=(FIX32_1 a, FIX32_1 b);
friend bool operator>=(FIX32_1 a, FIX32_1 b);
friend bool operator==(FIX32_1 a, FIX32_1 b);
friend bool operator!=(FIX32_1 a, FIX32_1 b);

volatile FIX32_1& operator<=(int b) volatile;
FIX32_1& operator<=(int b);
volatile FIX32_1& operator>=(int b) volatile;

```

Fig. 42

```

        FIX32_1& operator>>=(int b);

volatile FIX32_1& operator*=(FIX32_1 b) volatile;
        FIX32_1& operator*=(FIX32_1 b);
volatile FIX32_1& operator*=(int b) volatile;
        FIX32_1& operator*=(int b);
volatile FIX32_1& operator*=(float b) volatile;
        FIX32_1& operator*=(float b);
volatile FIX32_1& operator*=(double b) volatile;
        FIX32_1& operator*=(double b);

volatile FIX32_1& operator/=(FIX32_1 b) volatile;
        FIX32_1& operator/=(FIX32_1 b);
volatile FIX32_1& operator/=(int b) volatile;
        FIX32_1& operator/=(int b);
volatile FIX32_1& operator/=(float b) volatile;
        FIX32_1& operator/=(float b);
volatile FIX32_1& operator/=(double b) volatile;
        FIX32_1& operator/=(double b);

volatile FIX32_1& operator+=(FIX32_1 b) volatile;
        FIX32_1& operator+=(FIX32_1 b);
volatile FIX32_1& operator-=(FIX32_1 b) volatile;
        FIX32_1& operator-=(FIX32_1 b);

long value() {return val;}

// Other functions
friend FIX32_1 _fix321(long a);
friend long _bptn(FIX32_1 a);
friend float _float(FIX32_1 a);
friend double _double(FIX32_1 a);

friend FIX32_1 _abs(FIX32_1 a);
friend FIX32_1 _max(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _min(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _adds(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _subs(FIX32_1 a, FIX32_1 b);
friend int _bcnt1(FIX32_1 a);
friend int _bseq(FIX32_1 a);
friend int _bseq0(FIX32_1 a);
friend int _bseq1(FIX32_1 a);
friend FIX16_1 _round(FIX32_1 a);

friend int _extr(FIX32_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_1 a, unsigned int b, unsigned int c);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);

friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

```

Fig. 43

```

friend void    _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, int b);
friend FIX32_1 _div(FIX32_1 a, float b);
friend FIX32_1 _div(FIX32_1 a, double b);

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                     Member of FIX16_2
//
//                                     val : Actual value in 16 bits
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class FIX16_2{
    short val;
public:
    // constructor
    FIX16_2() {}
    FIX16_2(int a);
    FIX16_2(float a);
    FIX16_2(double a);
    FIX16_2(FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(volatile FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(const FIX16_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_2& operator=(FIX16_2 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_2& operator=(FIX16_2 a)
    {
        val = a.val;
        return *this;
    }

    friend FIX16_2 operator+(FIX16_2 a);
    friend FIX16_2 operator-(FIX16_2 a);

    friend FIX16_2 operator+(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator-(FIX16_2 a, FIX16_2 b);

    friend FIX16_2 operator*(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator*(int a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, int b);
    friend FIX16_2 operator*(float a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, float b);
    friend FIX16_2 operator*(double a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, double b);

```

Fig. 44

```

friend FIX16_2 operator/(FIX16_2 a, FIX16_2 b);
friend FIX16_2 operator/(FIX16_2 a, int b);
friend FIX16_2 operator/(FIX16_2 a, float b);
friend FIX16_2 operator/(FIX16_2 a, double b);

friend FIX16_2 operator<<(FIX16_2 a, int b);
friend FIX16_2 operator>>(FIX16_2 a, int b);

friend bool operator<(FIX16_2 a, FIX16_2 b);
friend bool operator>(FIX16_2 a, FIX16_2 b);
friend bool operator<=(FIX16_2 a, FIX16_2 b);
friend bool operator>=(FIX16_2 a, FIX16_2 b);
friend bool operator==(FIX16_2 a, FIX16_2 b);
friend bool operator!=(FIX16_2 a, FIX16_2 b);

volatile FIX16_2& operator<<=(int b) volatile;
FIX16_2& operator<<=(int b);
volatile FIX16_2& operator>>=(int b) volatile;
FIX16_2& operator>>=(int b);

volatile FIX16_2& operator*=(FIX16_2 b) volatile;
FIX16_2& operator*=(FIX16_2 b);
volatile FIX16_2& operator*=(int b) volatile;
FIX16_2& operator*=(int b);
volatile FIX16_2& operator*=(float b) volatile;
FIX16_2& operator*=(float b);
volatile FIX16_2& operator*=(double b) volatile;
FIX16_2& operator*=(double b);

volatile FIX16_2& operator/=(FIX16_2 b) volatile;
FIX16_2& operator/=(FIX16_2 b);
volatile FIX16_2& operator/=(int b) volatile;
FIX16_2& operator/=(int b);
volatile FIX16_2& operator/=(float b) volatile;
FIX16_2& operator/=(float b);
volatile FIX16_2& operator/=(double b) volatile;
FIX16_2& operator/=(double b);

volatile FIX16_2& operator+=(FIX16_2 b) volatile;
FIX16_2& operator+=(FIX16_2 b);
volatile FIX16_2& operator-=(FIX16_2 b) volatile;
FIX16_2& operator-=(FIX16_2 b);

short value() {return val;}

// Other functions
friend FIX16_2 _fix162(short a);
friend short _bptn(FIX16_2 a);
friend FIX16_2 _fix162(FIX32_2 a);
friend float _float(FIX16_2 a);
friend double _double(FIX16_2 a);

friend FIX16_2 _abs(FIX16_2 a);
friend FIX16_2 _max(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _min(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _adds(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _subs(FIX16_2 a, FIX16_2 b);

```

Fig. 45

```

friend int      _bcnt1(FIX16_2 a);
friend int      _bseq(FIX16_2 a);
friend int      _bseq0(FIX16_2 a);
friend int      _bseq1(FIX16_2 a);
friend FIX16_2  _round(FIX32_2 a);

friend int _extr(FIX16_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_2 a, unsigned int b, unsigned int c);

friend void      _mul(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void      _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void      _msu(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void      _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend FIX16_2   _div(FIX16_2 a, FIX16_2 b);
friend FIX16_2   _div(FIX16_2 a, int b);
friend FIX16_2   _div(FIX16_2 a, float b);
friend FIX16_2   _div(FIX16_2 a, double b);

};

////////////////////////////////////
//                               Member of FIX32_2
//
//                               val : Actual value in 32 bits
//
////////////////////////////////////
class FIX32_2{
    long val;
public:
    // constructor
    FIX32_2() {}
    FIX32_2(int a);
    FIX32_2(float a);
    FIX32_2(double a);
    FIX32_2(FIX16_2 a);
    FIX32_2(FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(volatile FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(const FIX32_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX32_2& operator=(FIX32_2 a) volatile
    {

```

Fig. 46

```

        val = a.val;
        return *this;
    }
    FIX32_2& operator=(FIX32_2 a)
    {
        val = a.val;
        return *this;
    }
    friend FIX32_2 operator+(FIX32_2 a);
    friend FIX32_2 operator-(FIX32_2 a);

    friend FIX32_2 operator+(FIX32_2 a, FIX32_2 b);
    friend FIX32_2 operator-(FIX32_2 a, FIX32_2 b);

    friend FIX32_2 operator*(FIX32_2 a, FIX32_2 b);
    friend FIX32_2 operator*(int a, FIX32_2 b);
    friend FIX32_2 operator*(FIX32_2 a, int b);
    friend FIX32_2 operator*(float a, FIX32_2 b);
    friend FIX32_2 operator*(FIX32_2 a, float b);
    friend FIX32_2 operator*(double a, FIX32_2 b);
    friend FIX32_2 operator*(FIX32_2 a, double b);

    friend FIX32_2 operator/(FIX32_2 a, FIX32_2 b);
    friend FIX32_2 operator/(FIX32_2 a, int b);
    friend FIX32_2 operator/(FIX32_2 a, float b);
    friend FIX32_2 operator/(FIX32_2 a, double b);

    friend FIX32_2 operator<<(FIX32_2 a, int b);
    friend FIX32_2 operator>>(FIX32_2 a, int b);

    friend bool operator<(FIX32_2 a, FIX32_2 b);
    friend bool operator>(FIX32_2 a, FIX32_2 b);
    friend bool operator<=(FIX32_2 a, FIX32_2 b);
    friend bool operator>=(FIX32_2 a, FIX32_2 b);
    friend bool operator==(FIX32_2 a, FIX32_2 b);
    friend bool operator!=(FIX32_2 a, FIX32_2 b);

    volatile FIX32_2& operator<=(int b) volatile;
        FIX32_2& operator<=(int b);
    volatile FIX32_2& operator>=(int b) volatile;
        FIX32_2& operator>=(int b);

    volatile FIX32_2& operator*=(FIX32_2 b) volatile;
        FIX32_2& operator*=(FIX32_2 b);
    volatile FIX32_2& operator*=(int b) volatile;
        FIX32_2& operator*=(int b);
    volatile FIX32_2& operator*=(float b) volatile;
        FIX32_2& operator*=(float b);
    volatile FIX32_2& operator*=(double b) volatile;
        FIX32_2& operator*=(double b);

    volatile FIX32_2& operator/=(FIX32_2 b) volatile;
        FIX32_2& operator/=(FIX32_2 b);
    volatile FIX32_2& operator/=(int b) volatile;
        FIX32_2& operator/=(int b);
    volatile FIX32_2& operator/=(float b) volatile;
        FIX32_2& operator/=(float b);
    volatile FIX32_2& operator/=(double b) volatile;
        FIX32_2& operator/=(double b);

```



Fig. 47

```

volatile FIX32_2& operator+=(FIX32_2 b) volatile;
    FIX32_2& operator+=(FIX32_2 b);
volatile FIX32_2& operator-=(FIX32_2 b) volatile;
    FIX32_2& operator-=(FIX32_2 b);

long value() {return val;}

// Other functions
friend FIX32_2 _fix322(long a);
friend long _bptn(FIX32_2 a);
friend float _float(FIX32_2 a);
friend double _double(FIX32_2 a);

friend FIX32_2 _abs(FIX32_2 a);
friend FIX32_2 _max(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _min(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _adds(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _subs(FIX32_2 a, FIX32_2 b);
friend int _bcnt1(FIX32_2 a);
friend int _bseq(FIX32_2 a);
friend int _bseq0(FIX32_2 a);
friend int _bseq1(FIX32_2 a);
friend FIX16_2 _round(FIX32_2 a);

friend int _extr(FIX32_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_2 a, unsigned int b, unsigned int c);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);

friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _msu(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, int b);
friend FIX32_2 _div(FIX32_2 a, float b);
friend FIX32_2 _div(FIX32_2 a, double b);

};

#if defined(__AMMPCC__)
#pragma pack_struct_default
#endif //__AMMPCC__

// other functions
#if defined(__AMMPCC__)

#pragma _enable_asm_begin

static inline FIX16_1 _fix161(FIX32_1 a)
{
    FIX16_1 result;

```



Fig. 48

```
    asm(vr0 = a) {  
    asr  vr1, vr0, 16;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX16_2 _fix162(FIX32_2 a)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a) {  
    asr  vr1, vr0, 16;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX16_1 _fix161(short a)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a) {  
    mov  vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX16_2 _fix162(short a)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a) {  
    mov  vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX32_1 _fix321(long a)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a) {  
    mov  vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX32_2 _fix322(long a)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a) {  
    mov  vr1, vr0;  
    }(result = vr1);  
}
```

Fig. 49

```
    return result;
}

static inline short _bptn(FIX16_1 a)
{
    short result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline short _bptn(FIX16_2 a)
{
    short result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline long _bptn(FIX32_1 a)
{
    long result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline long _bptn(FIX32_2 a)
{
    long result;

    asm(vr0 = a) {
        mov vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _abs(FIX16_1 a)
{
    FIX16_1 result;

    asm(vr0 = a) {
        absvh vr1, vr0;
    } (result = vr1);

    return result;
}
```

Fig. 50

```
static inline FIX32_1 _abs(FIX32_1 a)
{
    FIX32_1 result;

    asm(vr0 = a) {
        absvw vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_2 _abs(FIX16_2 a)
{
    FIX16_2 result;

    asm(vr0 = a) {
        absvh vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX32_2 _abs(FIX32_2 a)
{
    FIX32_2 result;

    asm(vr0 = a) {
        absvw vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _max(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b) {
        max vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX32_1 _max(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        max vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX16_2 _max(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;
```

Fig. 51

```
asm(vr0 = a, vr1 = b) {  
    max vr2, vr0, vr1;  
} (result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _max(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        max vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX16_1 _min(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX32_1 _min(FIX32_1 a, FIX32_1 b)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX16_2 _min(FIX16_2 a, FIX16_2 b)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    } (result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _min(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        min vr2, vr0, vr1;  
    } (result = vr2);  
}
```

Fig. 52

```
    }(result = vr2);  
    return result;  
}  
  
static inline FIX16_1 _adds(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_1 _adds(FIX32_1 a, FIX32_1 b)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_2 _adds(FIX16_2 a, FIX16_2 b)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _adds(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b) {  
        adds    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_1 _subs(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b) {  
        subs    vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}
```

Fig. 53

```
static inline FIX32_1 _subs(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        subs    vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX16_2 _subs(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        subs    vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline FIX32_2 _subs(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        subs    vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline int _bcnt1(FIX16_1 a)
{
    int result;

    asm(vr0 = a) {
        bcnt1    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bcnt1(FIX16_2 a)
{
    int result;

    asm(vr0 = a) {
        bcnt1    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bcnt1(FIX32_1 a)
{
    int result;
```

Fig. 54

```
    asm(vr0 = a) {  
        bcnt1    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bcnt1(FIX32_2 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bcnt1    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq     vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq0(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq0    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq1(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq1    vr1, vr0;  
    } (result = vr1);  
  
    return result;  
}  
  
static inline int _bseq(FIX32_1 a)  
{  
    int result;  
  
    asm(vr0 = a) {  
        bseq     vr1, vr0;  
    } (result = vr1);  
}
```

Fig. 55

```
    return result;
}

static inline int _bseq0(FIX32_1 a)
{
    int result;

    asm(vr0 = a) {
        bseq0    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq1(FIX32_1 a)
{
    int result;

    asm(vr0 = a) {
        bseq1    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq(FIX16_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq     vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq0(FIX16_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq0    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq1(FIX16_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq1    vr1, vr0;
    } (result = vr1);

    return result;
}
```



Fig. 56

```

static inline int _bseq(FIX32_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq    vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq0(FIX32_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq0   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int _bseq1(FIX32_2 a)
{
    int result;

    asm(vr0 = a) {
        bseq1   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_1 _round(FIX32_1 a)
{
    FIX16_1 result;

    asm(vr0 = a) {
        rndvh   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline FIX16_2 _round(FIX32_2 a)
{
    FIX16_2 result;

    asm(vr0 = a) {
        rndvh   vr1, vr0;
    } (result = vr1);

    return result;
}

static inline void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0, vr2, vr0, vr1;
    }
}

```

Fig. 57

```

    } (c = vr2);
}

static inline void _mulr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0, vr2, vr1, vr0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

Fig. 58

```

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0, vr2, vr1, vr0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mul(long &mh, long&ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {

```

Fig. 59

```

    fmachww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmacww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmacww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _macr(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmachhr m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

Fig. 60

```

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhww m0, vr2, vr1, vr0, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmsuww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmsuww m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msur(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhhr m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

inline FIX16_1 operator/(FIX16_1 a, FIX16_1 b)
{

```

Fig. 61

```

FIX16_1 result;

asm(vr0 = a, vr1 = b) {
    extw m0, vr3, vr0;
    aslp m0, vr4, mh0, vr3, 15;
    div  mh1, vr5, mh0, vr4, vr1;
    sath vr2, vr5;
} (result = vr2);

return result;
}

inline FIX16_2 operator/(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        aslp m0, vr4, mh0, vr3, 14;
        div  mh1, vr5, mh0, vr4, vr1;
        sath vr2, vr5;
    } (result = vr2);

    return result;
}

inline FIX32_1 operator/(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        mskgen vr3, 31, 31; //vr3 = 0x80000000
        cmpeq  C0:C1, vr1, vr3;
        [C0]   negvw vr2, vr0; //in the case of vr1==(-1), sign-change
        cmpgtn C2:C3, vr0, 0, C1; //in the case of vr1!=(-1)
        [C2]   negvw vr4, vr0; //sign-change vr0 negatively (vr0')
        [C3]   mov vr4, vr0;
        cmpgtn C2:C3, vr1, 0, C1;
        [C2]   negvw vr5, vr1; //sign-change vr1 negatively (vr1')
        [C3]   mov vr5, vr1;
        cmplen C2:C3, vr4, vr5, C1;
        [C2]   lsr vr6, vr0, 31; //vr0' <= 2vr1' in the case of (ovf decision),
over flow
        [C2]   lsr vr7, vr1, 31;
        [C2]   xor vr8, vr6, vr7;
        cmpeqn C4:C5, vr8, 0, C2;
        [C4]   mskgen vr2, 30, 0; //vr2 = 0x7fffffff
        [C5]   mskgen vr2, 31, 31; //vr2 = 0x80000000
        [C3]   extw m0, vr9, vr0; //in the case of dividend<divider
        [C3]   aslp m0, vr10, mh0, vr9, 31;
        [C3]   div mh1, vr2, mh0, vr10, vr1;
    } (result = vr2);

    return result;
}

```

Fig. 62

```

inline FIX32_2 operator/(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen vr3, 31, 31; //vr3 = 0x80000000
        mskgen vr4, 31, 30; //vr4 = 0xc0000000
        cmpgt C0:C1, vr0, 0;
        [C0] negvw vr5, vr0; //sign-change vr0 negatively (vr0')
        [C1] mov vr5, vr0;
        cmpgt C0:C1, vr1, 0;
        [C0] negvw vr6, vr1; //sign-change vr1 negatively (vr1')
        [C1] mov vr6, vr1;
        cmpeq C0:C1, vr0, vr3;
        cmplt C0:C1, vr6, vr4, C0; //vr0=-2 and vr1' < 0xc0000000?
        [C1] aslvw vr6, vr6, 1; //!when (vr0=-2 and vr1' < 0xc0000000)
        cmplea C2:C3, vr5, vr6, C1;
        [C2] lsr vr7, vr0, 31; //!in the case of (vr0=-2 and vr1'
        [C2] lsr vr8, vr1, 31; //< 0xc0000000) and vr0' <= 2vr1'
        [C2] xor vr9, vr7, vr8; //(ovf decision), overflow
        cmpeqn C4:C5, vr9, 0, C2;
        [C4] mskgen vr2, 30, 0; //vr2 = 0x7fffffff
        [C5] mskgen vr2, 31, 31; //vr2 = 0x80000000
        [C3] extw m0, vr10, vr0; //other than that
        [C3] aslp m0, vr11, m0, vr10, 30;
        [C3] div mh1, vr2, m0, vr11, vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, int b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        div mh1, vr4, m0, vr3, vr1;
        sath vr2, vr4;
    } (result = vr2);

    return result;
}

inline FIX16_2 operator/(FIX16_2 a, int b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        extw m0, vr3, vr0;
        div mh1, vr4, m0, vr3, vr1;
        sath vr2, vr4;
    } (result = vr2);

    return result;
}

```

Fig. 63

```

inline FIX32_1 operator/(FIX32_1 a, int b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        cmpeq    C0:C1, vr1, 1;
        [C0]    mov     vr2, vr0;    //when dividend=1, Rc = Ra
        [C1]    mskgen  vr3, 31, 31; //0x80000000
                cmpeqn  C2:C3, vr0, vr3, C1;
                cmpeqa  C4:C5, vr1, -1, C2;
        [C4]    mskgen  vr2, 30, 0;  //when dividend=-1 and divisor=-1, Rc =
0x7fffffff
        [C5]    extw    m0, vr4, vr0; //other than that
        [C5]    div     mh1, vr2, mh0, vr4, vr1;
    } (result = vr2);

    return result;
}

inline FIX32_2 operator/(FIX32_2 a, int b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen  vr3, 31, 31; //Rd = 0x80000000
        mskgen  vr4, 31, 30; //Re = 0xc0000000
        cmpgt   C0:C1, vr0, 0;
        [C0]    negvw   vr5, vr0;    //sign-change Ra negatively (Ra')
        [C1]    mov     vr5, vr0;
        aslww   vr6, vr1, 30; //int-->FIX32_2
        cmpgt   C0:C1, vr6, 0;
        [C0]    negvw   vr7, vr6;    //sign-change Rb negatively (Rb')
        [C1]    mov     vr7, vr6;
        cmpeq   C0:C1, vr0, vr3;
        cmplt   C0:C1, vr7, vr4, C0; //Ra=-2 and Rb' < 0xc0000000?
        [C1]    aslww   vr7, vr7, 1;  //!when(Ra=-2 and Rb' < 0xc0000000)
        cmplea  C2:C3, vr5, vr7, C1;
        [C2]    lsr     vr8, vr0, 31; //!in the case of (Ra=-2 and Rb' < 0xc0000000)
        [C2]    lsr     vr9, vr1, 31; // and Ra' <= 2Rb' (ovf decision)
        [C2]    xor     vr10, vr8, vr9; //overflow
        cmpeqn  C4:C5, vr10, 0, C2;
        [C4]    mskgen  vr2, 30, 0;  //Rc = 0x7fffffff
        [C5]    mskgen  vr2, 31, 31; //Rc = 0x80000000
        [C3]    extw    m0, vr11, vr0; //other than that
        [C3]    div     mh1, vr2, mh0, vr11, vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, float b)
{
    FIX16_1 c = b;
    return a/c;
}

```



Fig. 64

```
inline FIX16_1 operator/(FIX16_1 a, double b)
{
    FIX16_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, float b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, double b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, float b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, double b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, float b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, double b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX16_1& FIX16_1::operator/=(FIX16_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(FIX16_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}
```

Fig. 65

```
inline volatile FIX16_1& FIX16_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(FIX32_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(FIX32_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}
```

Fig. 66

```
}  
  
inline volatile FIX32_1& FIX32_1::operator/=(float b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline FIX32_1& FIX32_1::operator/=(double b)  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline volatile FIX32_1& FIX32_1::operator/=(double b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline FIX16_2& FIX16_2::operator/=(FIX16_2 b)  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline volatile FIX16_2& FIX16_2::operator/=(FIX16_2 b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline FIX16_2& FIX16_2::operator/=(int b)  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline volatile FIX16_2& FIX16_2::operator/=(int b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline FIX16_2& FIX16_2::operator/=(float b)  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline volatile FIX16_2& FIX16_2::operator/=(float b) volatile  
{  
    *this = *this / b;  
    return *this;  
}  
  
inline FIX16_2& FIX16_2::operator/=(double b)  
{  
    *this = *this / b;
```

Fig. 67

```
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(FIX32_2 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(FIX32_2 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

#pragma _enable_asm_end
```

Fig. 68

```
#endif __AMMPCC__
```

```
#endif __FIXAMMP__
```

Fig. 69

```

/*****
*
* (C) Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*     funcamp.h
*     Version:
*     Release:
*     Date:      2002/6/19
*
*****/

/* Avoid overloading */
#ifndef __FUNCAMP__
#define __FUNCAMP__

#include <stdlib.h>

#ifndef __AMMPCC__

long _abs(long);
long _max(long, long);
long _min(long, long);
long _adds(long, long);
long _subs(long, long);
int _bcnt1(long);
int _bseq0(long);
int _bseq1(long);
int _bseq(long);
int _log2(size_t size);
long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
void _clrm(long&, long&);
void _mul(long&, long&, long&, long, long);
void _mac(long&, long&, long&, long, long);
void _msu(long&, long&, long&, long, long);
void *_modulo_add(void *, int, int, size_t, void *);
void *_brev_add(void *, int, int, int, size_t, void *);

#else /* defined __AMMPCC__ */

#pragma _enable_asm_begin
#pragma _enable_inline_begin

long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
int _log2(size_t size);

static inline long
_abs(long data)
{
    long result;

    asm(vr0 = data) {
        abs    vr1, vr0;
    } (result = vr1);

    return result;
}

```

Fig. 70

```
static inline long
_max(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        max    vr2, vr1, vr0;
    } (result = vr2);

    return result;
}

static inline long
_min(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        min    vr2, vr1, vr0;
    } (result = vr2);

    return result;
}

static inline long
_adds(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        adds   vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline long
_subs(long data1, long data2)
{
    long result;

    asm(vr0 = data1, vr1 = data2) {
        subs   vr2, vr0, vr1;
    } (result = vr2);

    return result;
}

static inline int
_bcnt1(long data)
{
    long result;

    asm(vr0 = data) {
        bcnt1  vr1, vr0;
    } (result = vr1);

    return result;
}
```

Fig. 71

```

static inline int
_bseq0(long data)
{
    long result;

    asm(vr0 = data) {
        bseq0 vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int
_bseq1(long data)
{
    long result;

    asm(vr0 = data) {
        bseq1 vr1, vr0;
    } (result = vr1);

    return result;
}

static inline int
_bseq(long data)
{
    long result;

    asm(vr0 = data) {
        bseq vr1, vr0;
    } (result = vr1);

    return result;
}

static inline void
_clrm(long &mh, long &ml)
{
    asm() {
        mul m0, vr1, vr0, 0;
    } (mh = mh0, ml = ml0);
}

static inline void
_mul(long &mh, long &ml, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b) {
        mul m0, vr2, vr0, vr1;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void
_mac(long &mh, long &ml, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        mac m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```



Fig. 72

```

}

static inline void
_msu(long &mh, long &ml, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        msu m0, vr2, vr0, vr1, m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void *
_modulo_add(void *addr, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = imm << tmp1;

    asm(vr0 = addr, vr2 = base, vr3 = tmp2, vr4 = tmp3) {
        mov    CFR0, vr3;
        add    vr6, vr0, vr4;
        addmsk vr7, vr2, vr6;
    } (p = vr7);

    return p;
}

static inline void *
_brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3, tmp4;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = 16 - mask - tmp1;
    tmp4 = imm << tmp3;

    asm(vr0 = addr, vr1 = cnt, vr2 = base, vr3 = tmp2, vr4 = tmp3, vr5 = tmp4) {
        mov    CFR0, vr3;
        lsl    vr6, vr1, vr4;
        add    vr7, vr6, vr5;
        mskbrvh vr8, vr2, vr7;
    } (p = vr8);

    return p;
}

#pragma _enable_inline_end
#pragma _enable_asm_end

#endif /* __AMMPCC__ */

#endif /* __FUNCAMMP__ */

```

Fig. 73

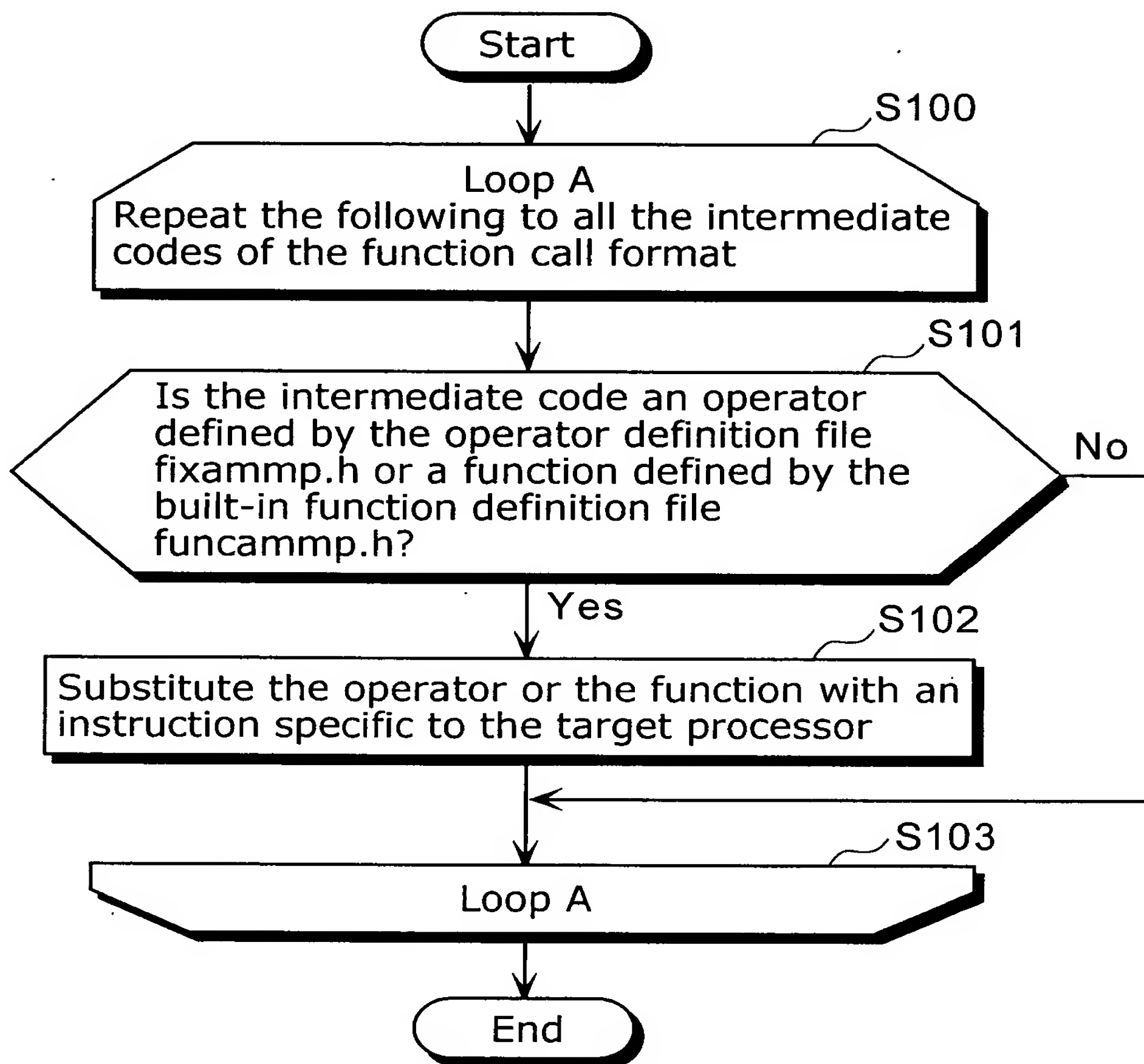


Fig. 74

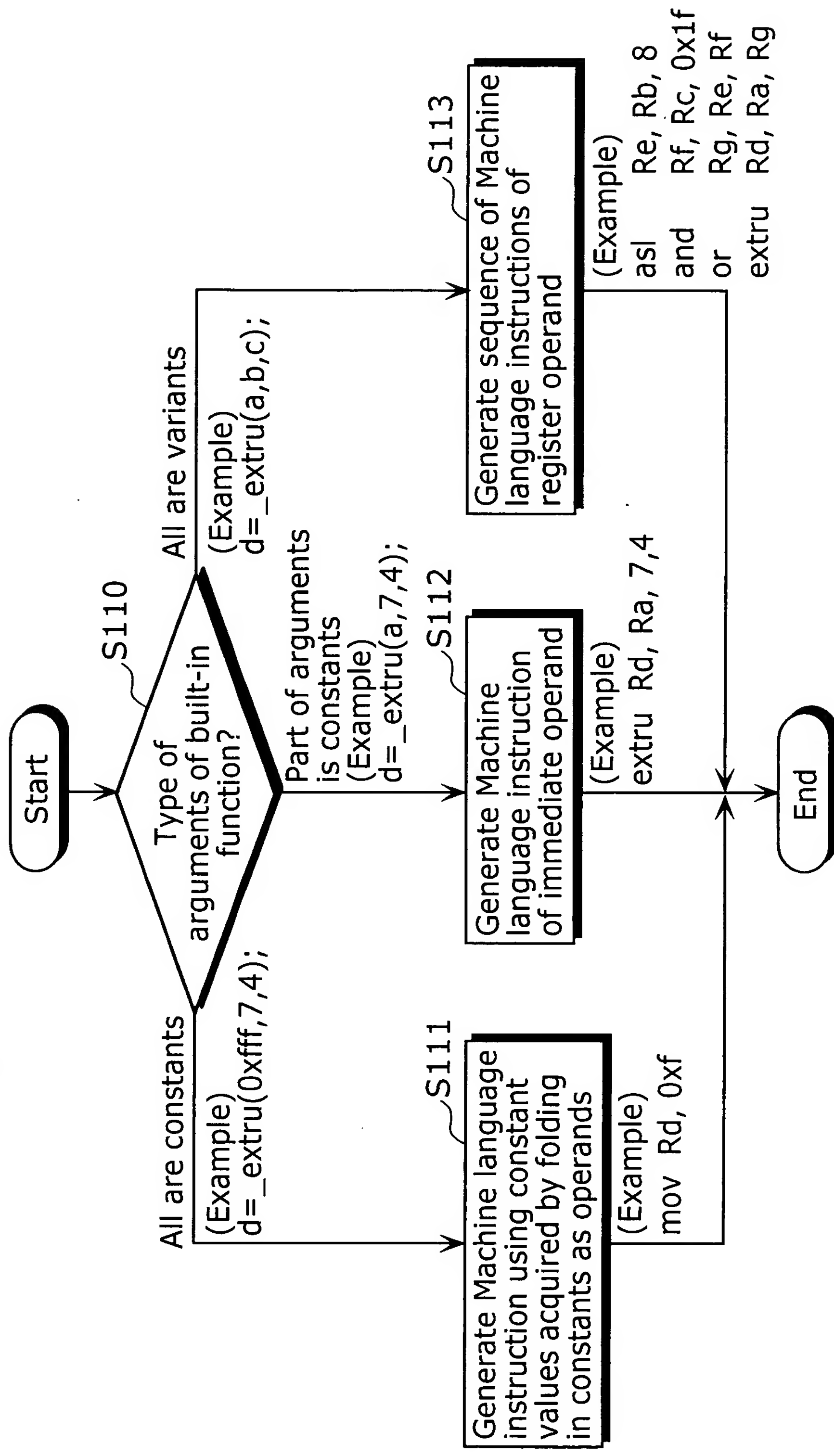


Fig. 75A

Ordinary arithmetic tree

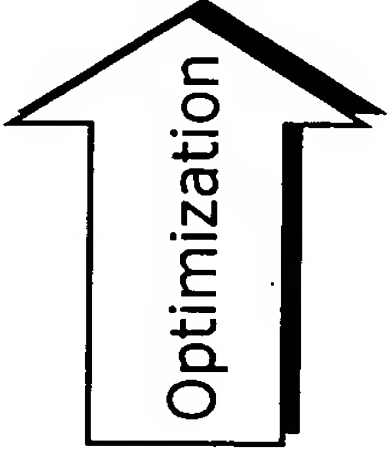
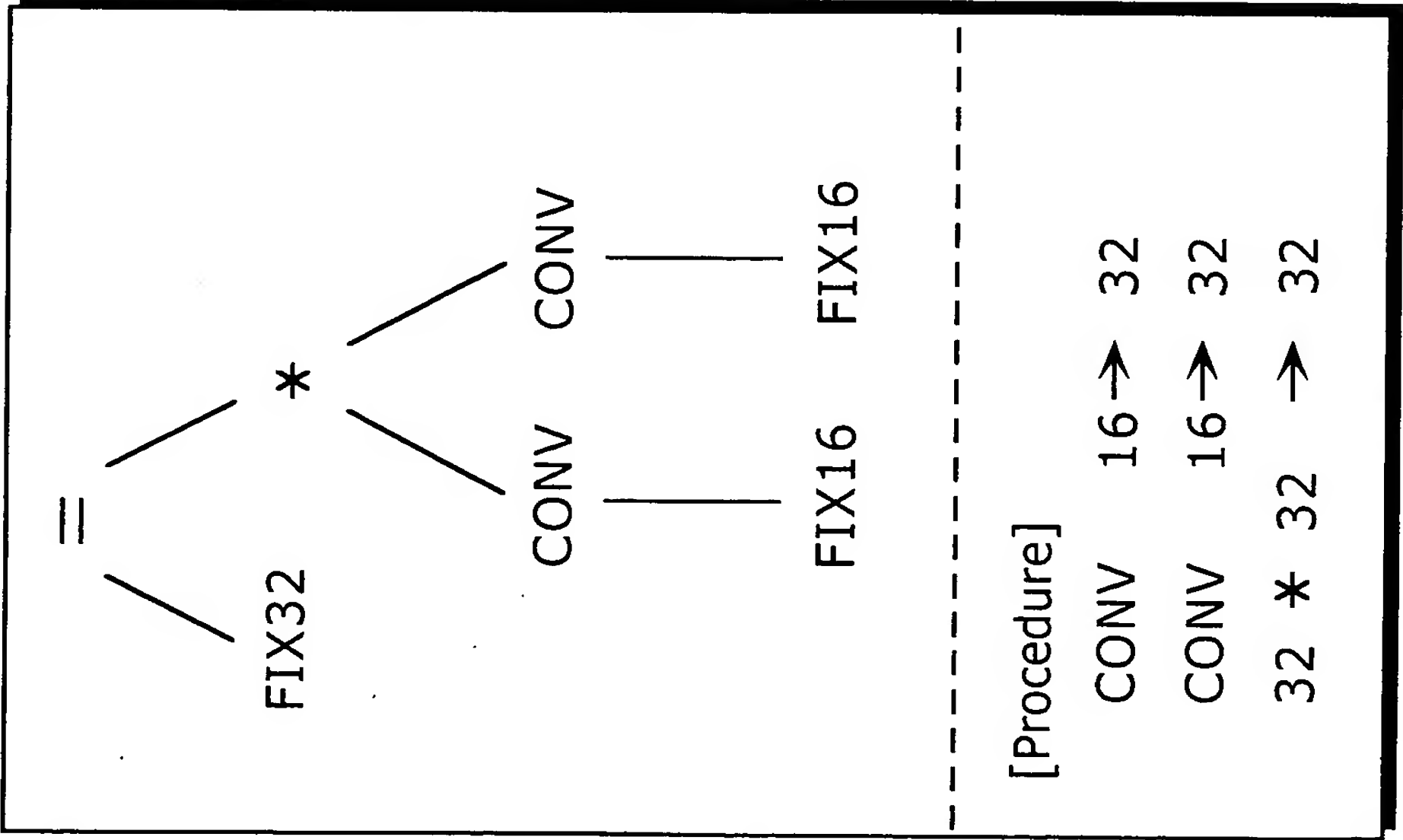


Fig. 75B

Arithmetic tree after optimization

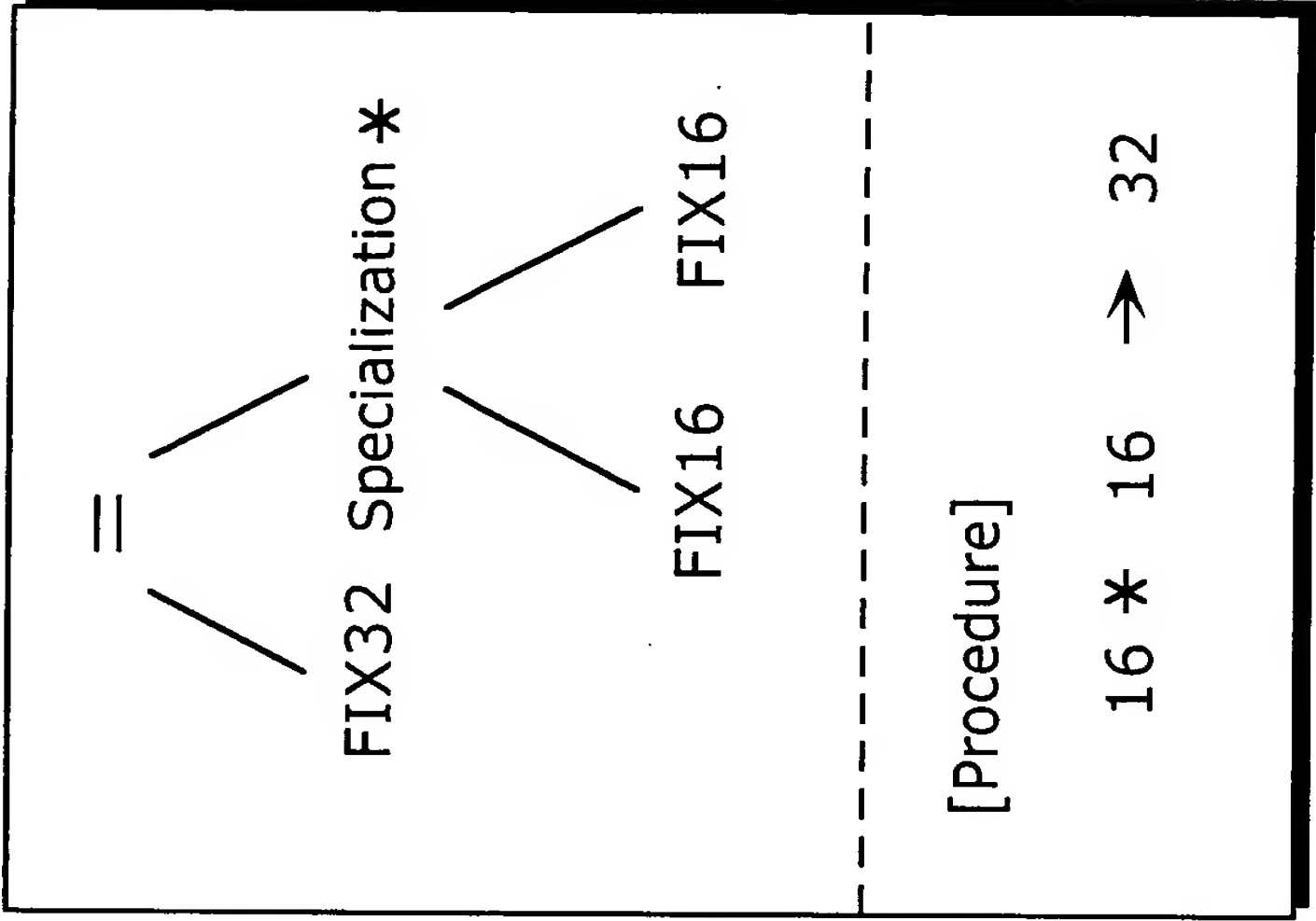


Fig. 76A

Example of configuring latency 1

<Example> Leave a space of 2 cycles between wte and rde

```
static inline int getbits(int a)
{
    int result ;
    asm (vr0 = a) {
        LATENCY L1, L2, 2 ;
        mov vr1, AVLD_BASEADDR ;
L1:      wte C0:C1, (vr1, AVLD_GETBITS), vr0 ;
L2:      rde C0:C1, vr2, (vr1, AVLD_READPORT) ;
    } (result = vr2) ;
    return result ;
}
```

Fig. 76B

Example of configuring latency 2

<Example> Leave a space of 2 cycles after wte

until next extended register access

```
static inline void skipbits(int a)
{
    asm (vr0 = a) {
        mov vr1, AVLD_BASEADDR ;
        wte C0:C1, (vr1, AVLD_SKIPBITS), vr0, LATENCY(2) ;
    } ;
}
```

Fig. 77A

Sample program

```
#pragma _save_fxpmode func
func(void)
{
    FIX16_1 a;
    (Main body)
}
```

Save FIX-type mode  
Configure FIX-type mode to \_1system  
→ (Main body)  
Return to FIX-type mode

Fig. 77B

Internal processing

```
Save:  mov vr0, PSR0
1Configuration: or  vr1, vr0, 0x20 + mov PSR0, vr1
0Configuration: andn vr1, vr0, 0x20 + mov PSR0, vr1
Return: mov PSR0, vr0
```

Fig. 77C

Application Example

Regarding four functions, f11, f21, f22 and f23, in the case of  
the function f11: \_1 system calling the function f21: \_2 system;  
the function f21: \_2 system calling the function f22: \_2 system;  
the function f22: \_2 system calling the function f23: \_2 system

---

Since the only function that may be called by other modes is f21,  
it is possible to switch to a normal mode by executing a pragma  
designation only to this function.

Fig. 78A

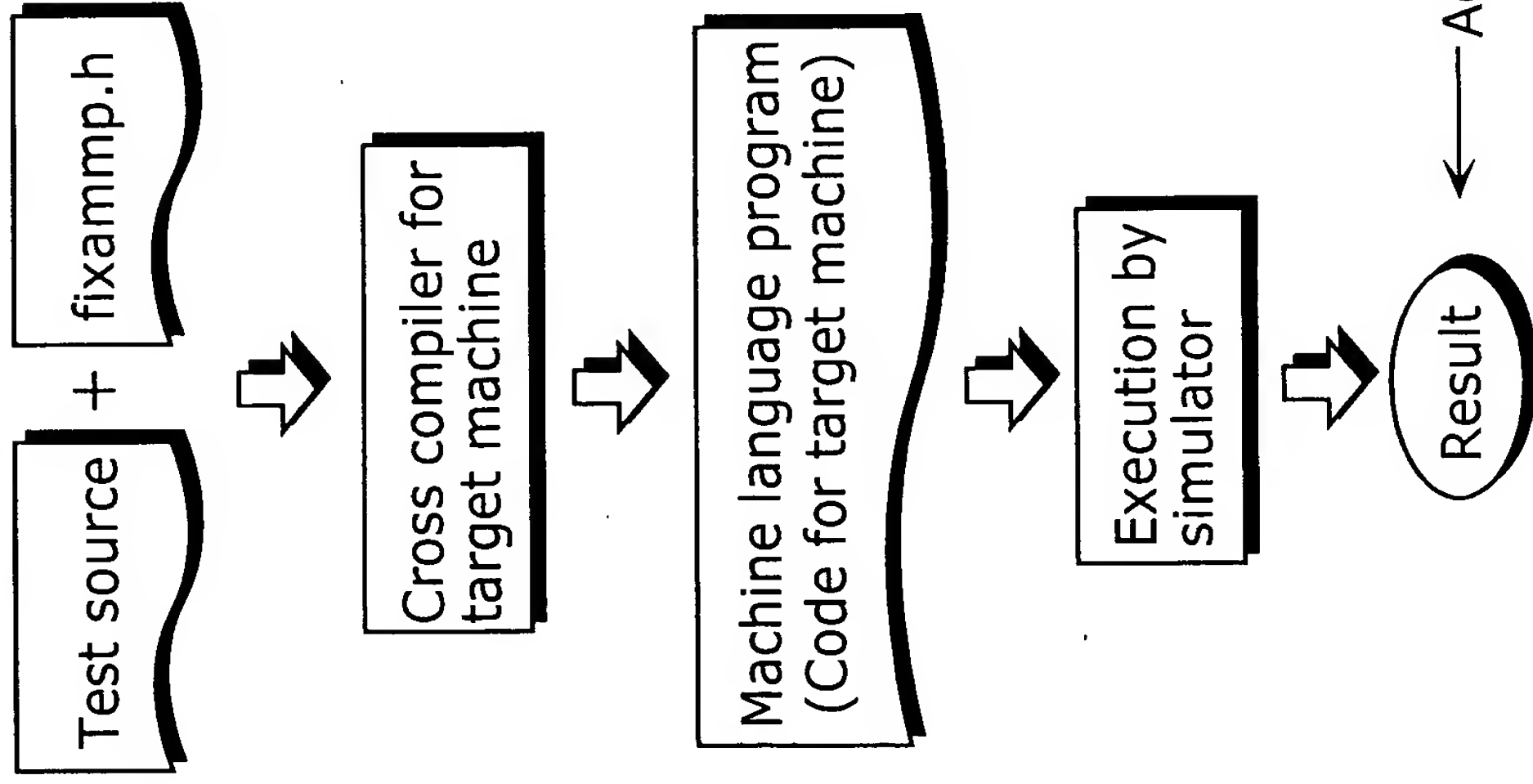


Fig. 78B

